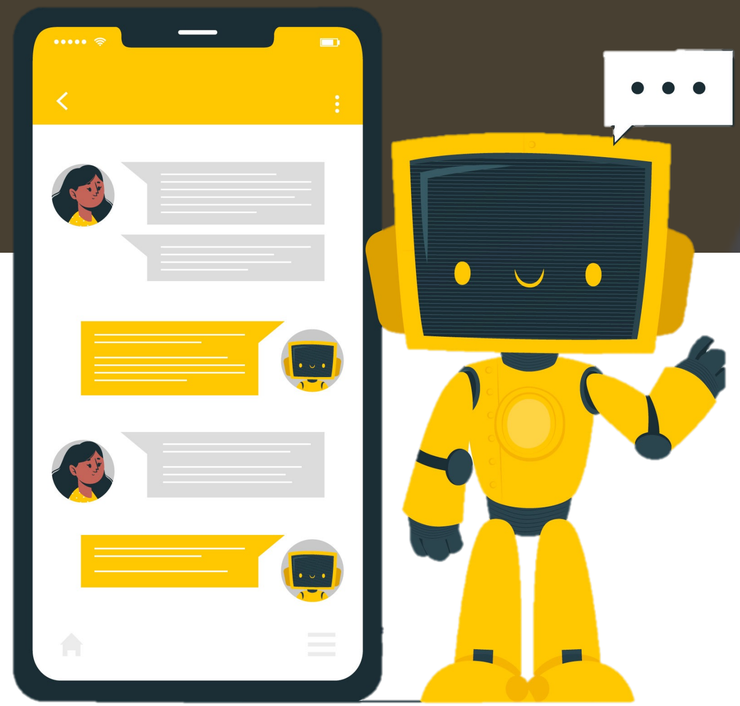
INVESTAI
WEALTHSIMPLE



VINCENT EL-GHOUBAIRA | XINGCHEN LUO
EMILY WU | CHIEN CHEN LIEW

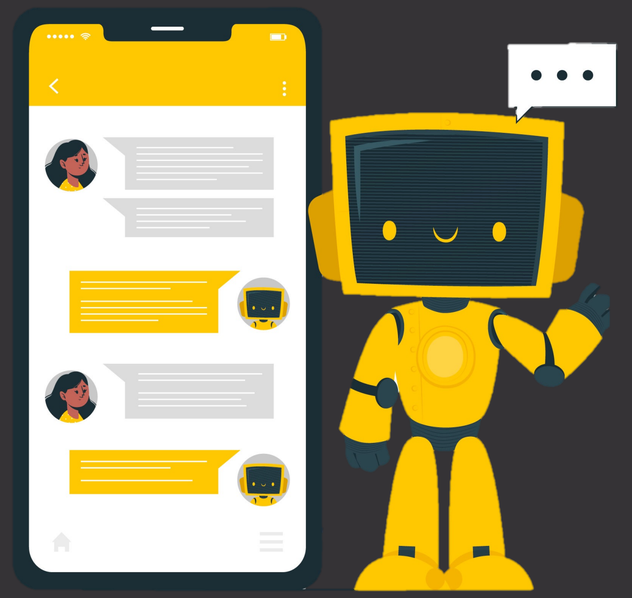
All of your investment,
made simple.

ABOUT US

Wealthsimple

- Founded in 2014
- Canadian FinTech
- Commission-Free Stock Trading
- No Trading Fees
- ETFs, Savings, Banking
- Breakdown Investment Barriers

Scenarios



Account Recommendations

Risk Profile

Investment Updates

ETFs Recommendations

Libraries and API

numpy/**numpy-financial**

The Major change

Set k=3 to find the 3 nearest neighbors to the customer's risk profile and intended returns.

Train and Test previous customers' data based on demographic information, return and risk levels and use current customer info to make recommendation

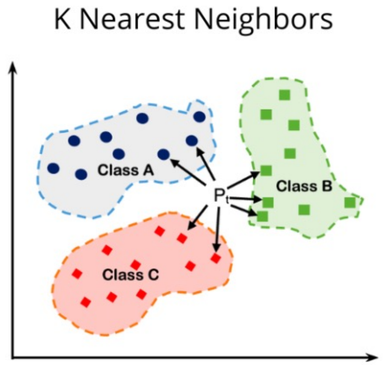
Normalized customer's risk profile & expected return

Normalized risk values and return of the ETFs

Filtered recommended ETFs for ETFs with Annual Returns greater than or equal to the expected returns, based on the customer's risk profile

Training set

Unsupervised learning



Proposal

Final Submission

Suggestions and Improvements

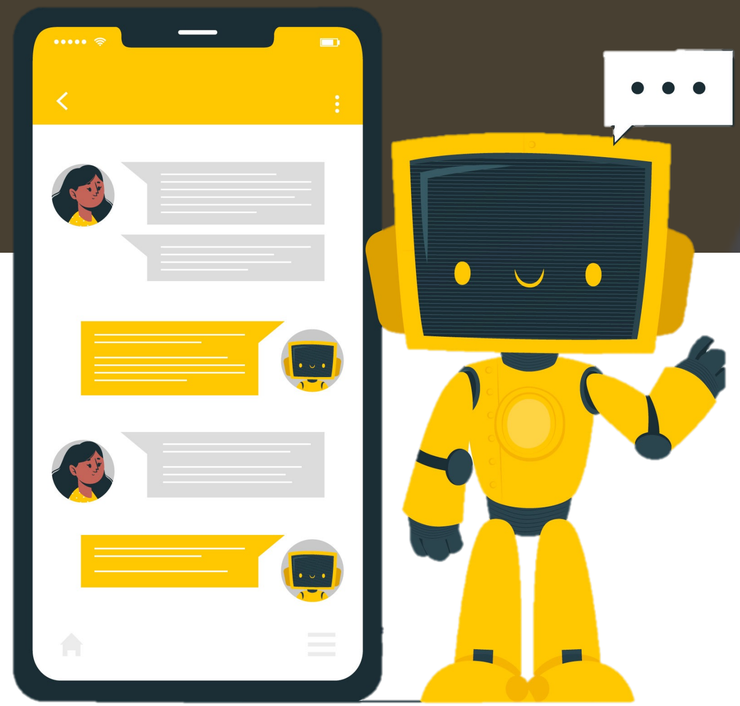
1. Expand the ETFs dataset to incorporate more ETFs/ other stock options for better suggestions. Currently, in some cases, our recommendation could return an empty value.

2. Explore ways to express the risk ranking for each ETF as a scale to allow better suggestions. For instance, could use standard deviation, Sharpe ratio, or R-square.

3. Explore integration with the app/ website interfaces for e.g. to allow users to create an account after knowing the types of accounts available in Scenario 1

4. If we could get real data from Wealth Simple, we could integrate better models and make better recommendations that fit new customers' profiles. (privacy…)

INVESTAI
THANK YOU

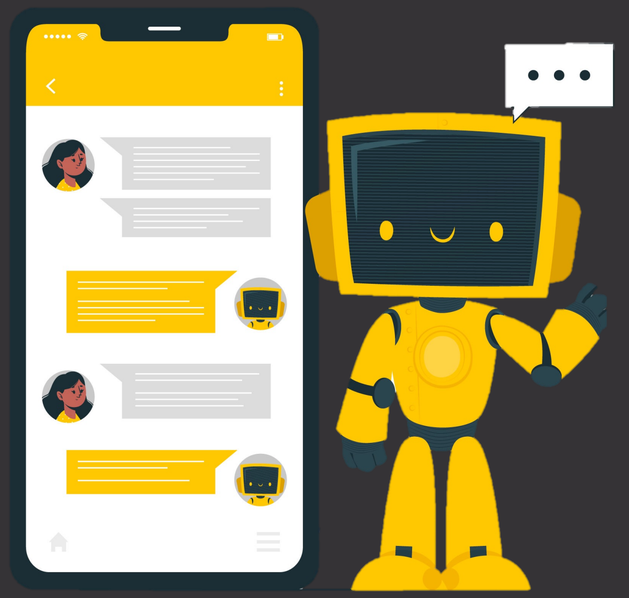


VINCENT EL-GHOUBAIRA | XINGCHEN LUO
EMILY WU | CHIEN CHEN LIEW

References

1. InvestAI image taken from: Image by storyset on Freepik
2. Yahoo finance for ETF screening
3. Wealthsimple
4. CFA Institute. (2022). Investment Risk Profiling: A Guide for Financial Advisors. Retrieved from https://www.cfainstitute.org/-/media/documents/survey/investment-risk-profiling.ashx
5. RBC Dominion Securities Inc. (2012). Client Risk Profile Questionnaire. Retrieved from https://ca.rbcwealthmanagement.com/documents/54275/54295/3.pdf/cd17c4ad-23a4-4767-8ada-781cbf50595d

Appendices - Codes



Account Recommendations

Risk Profile

Investment Updates

ETFs Recommendations

Scenario 1: Opening an Account

```
[32] class Scen1:
        def __init__(self):
            # Initialize a set of non-investment goals
            self.non_investment_goals = {1, 5, 6, 8}

        def get_customer_info(self):
            # Gather customer demographic information
            welcome_msg = input("Please provide us with your first name: ")
            age = int(input("What is your age? "))
            residency = input("Are you a Canadian resident? (y/n): ")
            annual_income = int(input(f"Hello {welcome_msg}! In order to figure out you
            return welcome_msg, age, residency, annual_income

        def validate_investment_options(self, age, residency, investment_goal):
            if age < 18 or residency.lower() != "y":
                return []

            # Check if the investment goal is a non-investment goal
            if investment_goal in self.non_investment_goals:
                # Return recommended options for non-investment goals
                if investment_goal == 1 or investment_goal == 5:
                    return ["GIC"]
                else:
                    return ["Consult with a financial manager."]
            else:
                valid_options = []

                # Recommend options based on specific investment goals
                if investment_goal == 2:
                    valid_options.append("FHSA")
                    valid_options.append("GIC")

                if investment_goal == 4:
                    valid_options.append("Brokerage Account")
                    valid_options.append("TFSA")
                    valid_options.append("GIC")

                if investment_goal == 7:
                    valid_options.append("RRSP")
                    valid_options.append("TFSA")
                    valid_options.append("GIC")

                return valid_options
```

```
    def run(self):
        # Greetings and gather customer information
        customer_name, customer_age, customer_residency, customer_income = self.get_customer_info()

        # Check if the customer is eligible for investment options
        if customer_age < 18 or customer_residency.lower() != "y":
            print("Sorry, you need to be at least 18 years old and a Canadian resident to open any type of account.")
        else:
            # Display investment goals
            goal = {
                1: "Build an emergency fund",
                2: "Buy a home",
                3: "Cover a child's college costs",
                4: "Achieve financial independence",
                5: "Create a budget",
                6: "Get debt under control",
                7: "Retire at 65",
                8: "Improve your credit score"
            }

            print("\nInvestment goals:")
            for key, value in goal.items():
                print(f"{key}: {value}")

            # Ask user to select an investment goal
            investment_goal = int(input("What is your investment goal? "))

            # Validate and recommend investment options
            valid_investment_options = self.validate_investment_options(customer_age, customer_residency, investment_goal)

            # Display recommended investment options
            if valid_investment_options:
                print("\nBased on your demographic information and investment goal, we recommend considering the following i
                for option in valid_investment_options:
                    print("- " + option)
                print("\nIf you need further assistance, don't hesitate to contact us on https://www.wealthsimple.com/en-ca/
            else:
                print("Sorry, we couldn't find any valid investment options for your selected goal and demographic informati
```

Scenario 2: Risk Profile

```python
class Scen2:
    def __init__(self):
        # Initialize attributes to store calculated rates and risk category
        self.total_score = None

    def get_input_score(self, question, options):
        # Get user input score for a question
        print(question)
        for index, option in enumerate(options):
            print(f"{index + 1}. {option}")

        while True:
            response = input("Enter your choice: ")
            if response.isdigit():
                choice = int(response)
                if 1 <= choice <= len(options):
                    return choice - 1
                else:
                    print("Please enter a valid choice.")
            else:
                print("Please enter a valid number.")

    def calculate_risk_category(self, total_score):
        # Calculate risk category based on total score
        if total_score <= 2:
            return "Extreme Low"
        elif total_score <= 5:
            return "Low"
        elif total_score <= 10:
            return "Below Average"
        elif total_score <= 15:
            return "Average"
        elif total_score <= 20:
            return "Above Average"
        else:
            return "High"

    def risk_rank(self,total_score):
```

```python
            return "High"

    def risk_rank(self,total_score):
        if total_score <= 2:
            return 0
        elif total_score <= 5:
            return 1
        elif total_score <= 10:
            return 2
        elif total_score <= 15:
            return 3
        elif total_score <= 20:
            return 4
        else:
            return 5

    def run(self):
        # Risk Profile Questionnaire
        questions = [
            "Question 1: How comfortable are you with the idea of taking risks?",
            "Question 2: What is your investment time horizon?",
            "Question 3: How much fluctuation in the value of your investments can you tolerate?",
            "Question 4: What is your familiarity with different investment options?",
            "Question 5: How important is it for you to preserve your initial investment?",
            "Question 6: How would you react if your investment value declines significantly in a short p
            "Question 7: What percentage of your investment portfolio are you willing to allocate to high
            "Question 8: How much effort are you willing to put into managing and monitoring your investm
            "Question 9: What is your reaction to the possibility of missing out on potentially high retu
            "Question 10: What is your reaction to the possibility of missing out on market gains due to
        ]

        options = [
            ["Not comfortable", "Slightly comfortable", "Moderately comfortable", "Very comfortable"],
            ["Less than 1 year", "1-5 years", "6-10 years", "More than 10 years"],
            ["Can't tolerate any loss", "Small loss is acceptable", "Moderate fluctuations are okay", "Si
            ["No idea about investment options", "Know a bit about common investments", "Decent understar
            ["Preserving initial investment is crucial", "Preservation is important, some risk can be tak
            ["Would panic and want to sell immediately", "Would feel uneasy but wait and watch", "Would h
            ["None", "Up to 20%", "Up to 50%", "More than 50%"],
            ["Set it and forget it", "Willing to spend some time occasionally", "Willing to actively mana
            ["Very upset", "Disappointed but understand", "Okay with it", "Understand that conservative c
            ["Regretful", "Fine with my choice", "Relieved", "Happy with my conservative approach"]
        ]
```

```python
        scores = []

        # Iterate through each question and get user input score
        for i, question in enumerate(questions):
            response = self.get_input_score(question, options[i])
            scores.append(response)

        # Calculate total score and determine risk category
        total_score = sum(scores)
        risk_category = self.calculate_risk_category(total_score)

        # Display the results
        print("\nTotal Score:", total_score)
        print("Risk Category:", risk_category)
        print("Risk Rank: ",self.risk_rank(total_score))
```

Scenario 3: Investment Update

```
[35] #create class Scenario 3 as alias scen3
     class Scen3:

         #initialize the attributes for the parameter self
         def __init__(self):
             self.user_data = {}

         def ask_question(self, question):
           # Function to ask a question and get user's input
             answer = input(question + " ")
             return answer

         # Function to fetch portfolio data using Yahoo Finance API
         def fetch_portfolio_data(self, symbols, period):

             data = yf.download(symbols, period=period)
             return data

         def plot_portfolio_data(self, portfolio_data, comparison_data=None):
             # Function to plot portfolio data
             #takes the earliest date
             earliest_date = portfolio_data.index[0]

             #to find the differences
             portfolio_pct_change = (portfolio_data['Close'] / portfolio_data.loc[earliest_date, 'Close'] - 1) * 100
             ax = portfolio_pct_change.plot(figsize=(10, 6), label=self.user_data['portfolio_name'])

             #plots the comparison data
             if comparison_data is not None:
                 comparison_pct_change = (comparison_data['Close'] / comparison_data.loc[earliest_date, 'Close'] - 1) * 100
                 comparison_pct_change.plot(ax=ax, linestyle='dotted', label="Comparison ETFs")

             plt.title("Portfolio and ETFs Performance (% Change from Earliest Date)")
             plt.xlabel("Date")
             plt.ylabel("Performance (%)")
             plt.legend()
             plt.show()

         def get_user_portfolios(self):
           # Function to retrieve user portfolios based on username and password
             username = self.user_data['username']
             password = self.user_data['password']
             user_portfolio_info = user_portfolios_df[(user_portfolios_df['username'] == username) & (user_portfolios_df['password'] == password)]
             #reset index for future retrievals
             user_portfolio_info.reset_index(inplace = True)

             if user_portfolio_info.empty:
                 print("Invalid username or password.")
                 return None

             return user_portfolio_info
```

```
         #to run all the respective functions in sequence by calling self.function
         def run(self):

             self.user_data['username'] = self.ask_question("1. What is your username?")
             self.user_data['password'] = self.ask_question("2. What is your password?")

             user_portfolios = self.get_user_portfolios()
             if user_portfolios is None:
                 return

             # Display available portfolios by iterating through all the rows of available portfolio
             print("Available portfolios:")
             for idx, row in user_portfolios.iterrows(): #for index and data (row):
                 print(f"{idx + 1}. {row['portfolio_name']}") #subsets for specific column of data

             # Select a portfolio
             while True:
                 portfolio_choice = int(self.ask_question("Select a portfolio (enter the number): "))
                 if 1 <= portfolio_choice <= len(user_portfolios):
                     break
                 else:
                     print("Invalid portfolio choice. Please enter a valid number.")

             selected_portfolio = user_portfolios.iloc[portfolio_choice - 1]

     # Store selected portfolio information
             self.user_data['portfolio_name'] = selected_portfolio['portfolio_name']
             self.user_data['portfolio'] = selected_portfolio['portfolio_symbols']
             self.user_data['period'] = self.ask_question("4. For which period would you like to check? (1mo/3mo/6mo/1y/5y/10y)")

             # Compare with other ETFs
             compare_choice = self.ask_question("5. Would you like to compare with other ETFs? (yes/no)")
             if compare_choice.lower() == 'yes':
                 self.user_data['compare'] = True
                 self.user_data['comparison_etfs'] = self.ask_question("6. Select which ETFs you would like to compare (comma-separated): ").split(',')
             else:
                 self.user_data['compare'] = False

     # Extract portfolio symbols and data period
             portfolio_symbols = selected_portfolio['portfolio_symbols'].split(',')
             period_mapping = {'1mo': '1mo', '3mo': '3mo', '6mo': '6mo', '1y': '1y', '5y': '5y', '10y': '10y'}
             data_period = period_mapping.get(self.user_data['period'], '1mo')

             portfolio_data = self.fetch_portfolio_data(portfolio_symbols, data_period)

     # Fetch comparison data if needed
             comparison_data = None
             if self.user_data['compare']:
                 comparison_symbols = self.user_data['comparison_etfs']
                 comparison_data = self.fetch_portfolio_data(comparison_symbols, data_period)

             print("\nPortfolio Data:")
             print(portfolio_data)

             self.plot_portfolio_data(portfolio_data, comparison_data)

             print("\nThank you! Here's the information you provided:")
             for key, value in self.user_data.items():
                 print(f"{key.capitalize()}: {value}")
```

Scenario 4: ETFs Recommendations

```python
def __init__(self):
    self.filtered_df = None
    self.scaler = None
    self.normalized_customer_risk = None
    self.recommended_ETFs = None
    self.recommendations = None


def financial_calculator(self, n, pmt, pv, fv):
    """Calculate and categorize the required rate of return."""
    from numpy_financial import rate
    import numpy_financial as npf

    rate_month = npf.rate(n, -pmt, -pv, fv) * 100
    rate_year = rate_month * 12

    required_rate_categories = [
        (5, "Very Low"),
        (15, "Low"),
        (25, "Average"),
        (35, "High"),
        (float("inf"), "Very High")
    ]

    required_rate = next(category for threshold, category in required_rate_categories if rate_year <= threshold)

    print("Your expected required rate per month:", round(rate_month, 2), "%")
    print("Your annual expected required rate:", round(rate_year, 2), "%")

    return rate_year
```

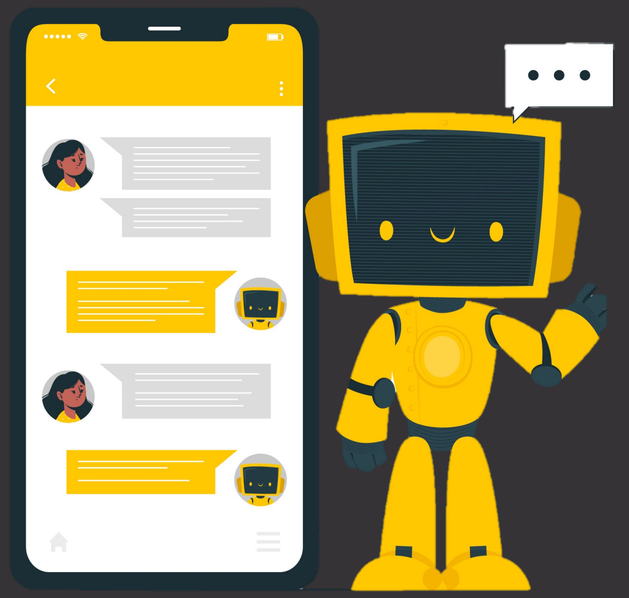
Gather Customer's Required rate of Return

```python
def gather_etf_data(self):
    """Gather ETF data from an external source."""
    excel_link = "https://onedrive.live.com/download.aspx?resid=6D3E3096AADFA7E5!20284&ithint=file%2cxlsx&wdo=2&auth
    df = pd.read_excel(excel_link, sheet_name="etf", engine="openpyxl")

    etfs_df = df["Symbol"]
    etfs_list = []
    returns_list = []

    for etf in etfs_df:
        #Fetch historical data using Yahoo Finance API
        ticker = yf.Ticker(etf)
        hist = ticker.history(period="5y")

        if not hist.empty:
            close_now = hist["Close"].iloc[-1]
            close_5y = hist["Close"].iloc[0]

            #calculate annual return
            annual_return = ((close_now / close_5y) ** (1 / 5)) - 1
            annual_return = annual_return * 100

            etfs_list.append(etf)
            returns_list.append(annual_return)
        else:
            etfs_list.append(etf)
            returns_list.append(None)

    #create DataFrame with ETF symbols and annual returns
    returns_df = pd.DataFrame({"Symbol": etfs_list, "Annual_Return": returns_list}).astype(
        {"Annual_Return": float}
    )

    #merge ETF data with annual returns
    merged_df = df.merge(returns_df, on="Symbol", how="left")
    merged_df.drop(["Returns", "Name"], axis=1, inplace=True)

    #map risk_mapping to numeric scale
    risk_mapping = {
        "low": 1,
        "below average": 2,
        "average": 3,
        "above average": 4,
        "high": 5
    }
    merged_df["Morningstar Risk "] = merged_df["Morningstar Risk "].str.lower().map(risk_mapping)

    return merged_df
```

Import ETFs: calculate risk and mapping the risk

```python
def recommend_etfs(self, merged_df, rate_year, risk_rank):
    """Recommend ETFs based on rate_year and risk_rank."""
    required_return = rate_year

    # Filter out ETFs with risk profile greater than or equal to customer's risk rank
    self.filtered_df = merged_df[merged_df["Morningstar Risk "] >= risk_rank]

    # Check if there are ETFs left after filtering by risk rank
    if self.filtered_df.empty:
        print("No ETFs meet the risk profile criteria.")
        return

    # Normalize risk values using Min-Max scaling
    self.scaler = MinMaxScaler()
    normalized_risk = self.scaler.fit_transform(self.filtered_df[["Morningstar Risk "]])

    # Normalize the customer's risk profile
    self.normalized_customer_risk = self.scaler.transform([[risk_rank]])

    n_neighbors = 3
    nbrs = NearestNeighbors(n_neighbors=n_neighbors, algorithm="auto").fit(normalized_risk)
    _, indices = nbrs.kneighbors(self.normalized_customer_risk)

    recommended_etfs = self.filtered_df.iloc[indices[0]]
    recommendations = recommended_etfs[recommended_etfs['Annual_Return'] >= rate_year]

    if recommendations.empty == True:
        print("\nNo ETFs recommended for your level of risk and/or returns. \nPlease revise your inputs or speak to

    else:
        recommendations.sort_values("Annual_Return",ascending = False,inplace=True) #sort by descending order
        print("\nRecommended ETFs:")
        print(recommendations.to_string(index=False)) #to remove the index when printing DataFrame

    return recommendations

def plot_recommendations(self, recommendations):
    """Plot the historical performance of recommended ETFs."""
    if recommendations is None or recommendations.empty:
        return

    plt.figure(figsize=(12, 6))

    #plot each ETF using closing data from Yahoo API for the past 5 years
    for symbol in recommendations['Symbol']:
        etf_data = yf.download(symbol, period="5y", progress=False)
        performance_ratio = (etf_data['Close'] / etf_data['Close'].iloc[0])*100  # Calculate performance ratio
        plt.plot(etf_data.index, performance_ratio, label=symbol)

    plt.xlabel('Date')
    plt.ylabel('ETF Performance (%)')
    plt.title('Recommended ETFs Historical Performance over the past 5 years')
    plt.legend()
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.show()

def run(self):
    n = int(input("How long is your investment horizon? (in terms of months) "))
    pv = float(input("How much money do you want to put in your account on day 1? "))
    pmt = float(input("How much money do you want to put in per month? (could be $0) "))
    fv = float(input("How much money do you want to make after your investment horizon? "))

    risk_rank = int(input("\nWhat was your risk ranking (1/2/3/4/5)? "))

    rate_year = self.financial_calculator(n, pmt, pv, fv)

    print(f"\nLooking for ETFs Recommendations with annual returns of {rate_year:.2f}% with risk ranking of {risk_ra

    merged_df = self.gather_etf_data()
    recommendations = self.recommend_etfs(merged_df, rate_year, risk_rank)

    self.plot_recommendations(recommendations)
```

Appendices - Codes



Account Recommendations

Risk Profile

Investment Updates

ETFs Recommendations

chatbot. The code handles user inputs, ensuring that valid options are selected and executed.

In [13]:
```
print("Hello! Welcome to InvestAI chatbot.")

while True:
    try:
        scenario = int(input("""
        What service would you like us to provide today:
        Account type recommendation: 1
        Risk level evaluation: 2
        Your Profile overview: 3
        ETF Recommendation: 4
        Exit: 0
        """))  # Convert input to integer

        if scenario == 0:
            print("Exiting chatbot...\nThank you for using InvestAI service! contact us on https://www.wealthsimple.
            break  # Exit the loop
        elif scenario in [1, 2, 3, 4]:
            if scenario == 1:
                chatbot = Scen1()
            elif scenario == 2:
                chatbot = Scen2()
            elif scenario == 3:
                chatbot = Scen3()
            elif scenario == 4:
                chatbot = Scen4()
            chatbot.run()
        else:
            print("Invalid scenario. Please select a valid scenario (1, 2, 3, 4) or type 0 to exit.")

    except ValueError:
        print("Invalid input. Please enter a valid number.")
```

```
What service would you like us to provide today:
Account type recommendation: 1
Risk level evaluation: 2
Your Profile overview: 3
ETF Recommendation: 4
Exit: 0
```

```
Hello! Welcome to InvestAI chatbot.
```