**Importance of Ethical AI In AML Solutions**:

The financial industry is not just undergoing change, it's on the brink of a radical transformation. Financial Institutions, FinTech, and RegTech companies are harnessing their expertise to build Artificial Intelligence (AI)-based Anti-Money Laundering (AML) solutions, paving the way for unprecedented opportunities to combat financial crime. This is a testament to the potential of AI in revolutionizing the industry.  
  
The primary focus of this solution is detecting and preventing money laundering activities. By utilizing enormous quantities of internal and external data, AI enables a more practical approach to identifying suspicious activities. This plays a crucial role in managing regulatory risks, avoiding penalties, effectively utilizing resources, and maintaining trust within the industry.  
  
However, unlocking the full potential of AI requires a commitment to ethical principles. Your expertise, as an industry expert, is crucial in shaping the future of AI in AML solutions. This survey aims to understand how you envision achieving a balance between efficiency and profitability alongside transparency, fairness, and trust in AI-powered AML solutions.

**QuantLib basics:**

**Date Class:**

The Date object can be created using the constructor as Date (day, month, year). It would be worthwhile to pay attention to the fact that day is the first argument, followed by month and then the year. This is different from the Python datetime object instantiation.

The fields of the Date object can be accessed using the month (), Day of Month () and year () methods. The weekday () method can be used to fetch the day of the week. The Date objects can also be used to perform arithmetic operations such as advancing by days, weeks, months etc. Periods such as weeks or months can be denoted using the Period class. Period object constructor signature is Period (num\_periods, period\_type). The num\_periods is an integer and represents the number of periods. The period\_type can be Weeks, Months and Years. One can also do logical operations using the Date object. The Date object is used in setting valuation dates, issuance and expiry dates of instruments. The Period object is used in setting tenors, such as that of coupon payments, or in constructing payment schedules.

**Calendar Class:**

The Date arithmetic above did not take holidays into account. But valuation of different securities would require taking into account the holidays observed in a specific exchange or country. The Calendar class implements this functionality for all the major exchanges.

The add Holiday and remove Holiday methods in the calendar can be used to add and remove holidays to the calendar respectively. If a calendar has any missing holidays or has a wrong holiday, then these methods come handy in fixing the errors. The businessDaysBetween method helps find out the number of business days between two dates per a given calendar. Let us use this method on the us\_calendar and italy\_calendar as a sanity check.

In valuation of certain deals, more than one calendar’s holidays are observed. QuantLib has JointCalendar class that allows you to combine the holidays of two or more calendars. Let us take a look at a working example.

**Schedule Class:**

The Schedule object is necessary in creating coupon schedules or call schedules. Schedule object constructors have the following signature.

Here we have generated a Schedule object that will contain dates between effective date and termination date with the tenor specifying the Period to be Monthly. The calendar object is used for determining holidays. Here we have chosen the convention to be the day following holidays. That is why we see that holidays are excluded in the list of dates. The Schedule class can handle generation of dates with irregularity in schedule. The two extra parameters first Date and nextToLastDate parameters along with a combination of forward or backward date generation rule can be used to generate short or long stub payments at the front or back end of the schedule. For example, the following combination of first Date and backward generation rule creates a short stub in the front on the January 15, 2015. Using the next To Last Date parameter along with the forward date generation rule creates a short stub at the back end of the schedule.

Using the backward generation rule along with the first Date allows us to create a long stub in the front. Below the first two dates are longer in duration than the rest of the dates.

Similarly, the usage of nextToLastDate parameter along with forward date generation rule can be used to generate long stub at the back of the schedule.

**Interest Rate:**

The Interest Rate class can be used to store the interest rate with the compounding type, day count and the frequency of compounding. Below we show how to create an interest rate of 5.0% compounded annually, using Actual/Actual day count convention.

Let’s say if you invest a dollar at the interest rate described by interest rate, the compound Factor method in the Interest Rate object gives you how much your investment will be worth after any period. Below we show that the value returned by compound factor for 2 years agrees with the expected compounding formula.

The discount Factor method returns the reciprocal of the compound Factor method. The discount factor is useful while calculating the present value of future cashflows. A given interest rate can be converted into other compounding types and compounding frequency using the equivalent Rate method.

The discount factor for the two Interest Rate objects, interest rate and new\_interest\_rate are the same, as shown below. The implied Rate method in the Interest Rate class takes compound factor to return the implied rate. The implied Rate method is a static method in the Interest Rate class and can be used without an instance of Interest Rate. Internally the equivalent Rate method invokes the implied Rate method in its calculations.