**TYPES OF SSRS/DOMO/TABLEAU REPORTING KPI’s**

**POINT IN TIME**

A report snapshot is a report that contains layout information and query results that were retrieved at a specific point in time.

**YTD/YOY**

**YoY/MoM/QoQ**

**Yearly, Monthly, Quarter**

YOY performance is frequently used by investors seeking to gauge whether a company's [financial performance](http://www.investopedia.com/terms/f/financialperformance.asp) is improving or worsening. 

Reasoning

YOY comparisons are popular when analyzing a company's performance because they help mitigate seasonality, a factor that can influence most businesses. Sales, profits and other financial metrics change during different periods of the year due to the fact most lines of business have peak and low demand seasons. For example, retailers have a peak demand season during the holiday shopping season, which falls in the fourth quarter of the year. To properly quantify a company's performance, it makes sense to compare revenue and profits year over year. The fourth-quarter performance in one year should be compared to the fourth-quarter performance in other years. If an investor looked at a retailer's results in the fourth quarter versus the prior third quarter, it may appear a company is undergoing unprecedented growth when it is actually seasonality that is influencing the difference in the results.

ALTER procedure [dbo].[reportYearOverClaimsTrendByPerils]

@StartingDate as datetime = null,

@EndingDate as datetime = null,

@AsOfDate as datetime = null,

@CompanyID as integer = null,

@LookBackPeriod as integer = null,

@ReportTypeID as integer = null

as

begin

set nocount on

if @EndingDate is null set @EndingDate = cast(dateadd(day, -day(getdate()), getdate()) as date)

if @StartingDate is null or @StartingDate > @EndingDate set @StartingDate = dbo.SpecialDate(@EndingDate, 'FOY')

if @AsOfDate is null set @AsOfDate = cast(dateadd(day, -1, getdate()) as date)

if @CompanyID not in (1, 2) set @CompanyID = null

if @ReportTypeID is null set @ReportTypeID = 1 -- 'Reported during the Period'

declare @OccurredOnly as bit = iif(@ReportTypeID = 2, 1, 0) -- 'Reported and Occurring during the Period'

if object\_id('tempdb.dbo.#Query') is not null drop table #Query

**begin with**

**DateRange as (**

**select 1 as DateRangeID,@StartingDate as StartingDate,**

**dateadd(millisecond, -3, dateadd(day, 1, @EndingDate)) as EndingDate,**

**dateadd(millisecond, -3, dateadd(day, 1, @AsOfDate)) as AsOfDate**

**union all**

**select dr.DateRangeID + 1 as DateRangeID,**

**dateadd(year, -1, dr.StartingDate) as StartingDate,**

**dateadd(year, -1, dr.EndingDate) as EndingDate,**

**dateadd(year, -1, dr.AsOfDate) as AsOfDate**

**from DateRange as dr**

**where dr.DateRangeID <= @LookBackPeriod**

**),**

**TRAILING 12 MONTHS**

So, while with trailing returns you would take just a 1 year period ending today. With rolling returns, you would take many such 1 year periods over the history of the funds with rolling intervals of 1 month, 3 month or 1 year.

//This code goes back a year

SELECT dateadd(month, -11, dateadd(day, -day(@AsOfDate) + 1, @AsOfDate))

**ROLLING 12 MONTHS**

SELECT sum(TRANSACTION\_AMOUNT)

FROM TRANSACTIONS

WHERE datediff(month, TRANSACTION\_DATE, getdate()) <= 12

AND TRANSACTION\_DATE <= getdate()

SET @StartDate = DATEADD(mm,-11,@EndDate)

SET @StartDate = DATEADD(dd,-(Day(@StartDate)-1),@StartDate)

SET @EndDate = DATEADD(SS,-1,DATEADD(DD,1,@EndDate))

**YTD/QTD/MTD/WTD**

SELECT sum(TRANSACTION\_AMOUNT)

FROM TRANSACTIONS

WHERE datediff(year, TRANSACTION\_DATE, getdate()) = 0

AND TRANSACTION\_DATE <= getdate()

declare @FirstOfYear as datetime,

@FirstOfMonth as datetime,

@FirstOfWeek as datetime

if @AsOfDate is null or @AsOfDate >= cast(getdate() as date)

set @AsofDate = dateadd(day, -1, cast(getdate() as date))

else

set @AsOfDate = cast(@AsOfDate as date)

select @FirstOfYear = dateadd(day, -datepart(dayofyear, @AsOfDate) + 1, @AsOfDate),

@FirstOfMonth = dateadd(day, -datepart(day, @AsOfDate) + 1, @AsOfDate),

@FirstOfWeek = dateadd(day, -datepart(dw, @AsOfDate) + 1, @AsOfDate),

@AsOfDate = dateadd(millisecond, -3, dateadd(day, 1, @AsOfDate))

**RETENTION**

### Retention

We’ll start with the most common metric: User retention. Simply put, we want to know how many users were active both last month and this month.

Let’s start by aggregating our **events** table into a monthly rollup, like so:

**with** monthly\_activity **as** (

**select** **distinct**

date\_trunc('month', created\_at) **as** **month**,

user\_id

**from** events

)

Now that we have this table, we’ll join it to itself. The left-hand-side of the join will represent this month, and the right-hand side will represent last month. We want to make sure that there’s only one row per **current\_month** per **user\_id**, and that row only gets included in the join if the user was active this month and last.

Here’s the query:

**with** monthly\_activity **as** (

**select** **distinct**

date\_trunc('month', created\_at) **as** **month**,

user\_id

**from** events

)

**select**

this\_month.**month**,

**count**(**distinct** user\_id)

**from** monthly\_activity this\_month

**join** monthly\_activity last\_month

**on** this\_month.user\_id **=** last\_month.user\_id

**and** this\_month.**month** **=** last\_month.**month** **+** interval '1 month'

**group** **by** **month**

Our two join conditions are:

1. **this\_month.month = last\_month.month + interval '1 month'**: This sets up how the join works. We want rows that include activity from this month and activity from last month, so we can reason about both time periods in the same statement.
2. **this\_month.user\_id = last\_month.user\_id**: Ensures one row per **user\_id**, and importantly, excludes rows where the user wasn’t present both months.

declare @PastYearAsOfDate as date = dateadd(day, -1, getdate())

select s.StateCode as 'State'

,COUNT(ly.PolicyID ) as 'Start'

,SUM(isnull(ly.PolicyID / ty.PolicyID, 0)) as 'Retained'

,COUNT(ly.PolicyID ) - SUM(isnull(ly.PolicyID / ty.PolicyID, 0)) as 'Diff'

, SUM(isnull(CAST(ly.PolicyID as money) / ty.PolicyID, 0)) / COUNT(ly.PolicyID)\*100 as '%Retained'

,isnull(left(a.ZipCode,5),'') as ZipCode

,@PastYearAsOfDate AS 'EffectiveDate'

,isnull(ag.AgencyName,'') as 'AgencyName'

,isnull(c.FIPS, '') as 'FIPS'

, isnull(c.CountyDescription,'') as 'County'

from InforcePolicy as ly

left join InforcePolicy as ty

on ty.PolicyID = ly.PolicyID

and @PastYearAsOfDate

between ty.EffectiveDate

and ty.ExpirationDate

Join Policy as p

ON ly.PolicyID = p.PolicyID

JOIN [dbo].[Agency] ag ON ag.AgencyID = p.AgencyID

join company cp on cp.CompanyID = p.CompanyID

join product pd on pd.ProductID = p.ProductID

join states s on s.StateID = p.StateID

join InsuredEntity ie on ie.PolicyID = p.PolicyID

join Addresses a on a.AddressId = ie.PropertyAddressID

left join county c on c.CountyID = a.CountyID

where dateadd(year, -1, @PastYearAsOfDate)

between ly.EffectiveDate

and ly.ExpirationDate

GROUP BY s.StateCode, isnull(ag.AgencyName,''),isnull(c.FIPS, ''), isnull(c.CountyDescription,''),isnull(left(a.ZipCode,5),'')

**Previous year as of 1 year ago from today to beginning of previous year**

**i.e. 1st 9 months of the year last year**

declare @ThisYear datetime = getdate()

declare @LastYear datetime = dateadd(year, -1, @ThisYear)

where p.CreatedDate

between dateadd(day, -datepart(dayofyear, @LastYear) + 1, cast(@LastYear as date))

and @LastYear

**Trending**

**Predictions**

**Aging report**