**Överlämning EKOSTAT-tool, Magnus Wenzer**

Hello hackers!

Jag har kommit en bra bit på vägen när det gäller utformningen av grundstrukturen i verktyget. Jag har försökt att följa strukturen som vi tittade på innan sommaren. Namnsättningen på klasser, objekt, metoder och resultat från beräkningar har jag dikterat efter den insikt jag har. Dessa behöver med all sannolikhet att ändras för bättre förståelse och tydlighet. OBS. Vi använder Python 3.

**Övergripande strukturen för klasser**

* Allt styrs från en **EventHandler** (ej skapad). Utformingen av denna ligger antagligen primärt på Arnolds bord och därför är den för närvarande beskriven i modulen *mw\_test\_file.py*. alternativt *quality\_factors.py*.
* EventHandler håller objekt av klassen **QualityFactor** som beskriver de olika kvalitetsfaktorerna.
* Varje kvalitetsfaktor använder en eller flera indikatorer (objekt av klassen **Indicator**) för att få fram status. Dessa initieras vid skapandet av kvalitesfaktorobjektet.
* Varje indikator använder i sin tur en eller flera parameterobjekt av klassen **Parameter** (alternativt **CalculatedParameter** om den är beräknad).
* All data, utom resulterande data i sista steget vid beräkning, hålls av objekt av klass **DataHandler**. Jag har hittills endast använt kolumndata (Fyskem) men jag tror att vi kom överens om att vi förmodligen även behöver raddat. Som koden är skriven nu har jag utgått ifrån all fyskemdata ligger i kolumnformat i en pandas.DataFrame. Jag har bara använt mig av en inputfil för att få data att jobba med. Tror att Johannes har bra koll på alla försteg som behöver göras får att få en homogen DataFrame från potentiellt olika källor. Jag har dålig koll på hopslagning av pd.df (append, join och merge mm.).
* **DataFilter** håller information om hur data ska filtreras. Filtrering görs genom att skicka ett DataFilter-objekt till ett DataHandler-objekt. I retur får man ett filtrerat DataHandler-objekt.
* **ToleranceFilter** används när man söker data i andra DataHandlers.
* **RefValues** håller information och metoder för att beräkna numerisk klassning. Exemplet som finns nu är från Örjans skript. Lena tittar på detta. RefValues är en singleton där varje parameter (ex. totn\_winter) är av klassen ParameterRefAreaType som i sin tur håller object av klassen RefTypeArea för varje typområde. Strukturen är: RefValues().par[type\_area] (exempel: RefValues()[’totn\_winter’][’22’])

Klasserna hålls i motsvarande modul (Indicators hålls i *indicators.py* osv.). Dessa moduler kan köras separat för att testa innehållet.

Utöver dessa moduler finns moduler som innehåller klasser utan klar roll, tex har jag förberett StationList, ParameterList (används i nuläget) samt hjälpfunktioner som tex lat\_lon\_dist().

**Exempel: Schematiskt flöde för beräkning av kvalitetsfaktor NP**

Här följer en beskrivning om hur jag tänkt flödet för att beräkna kvalitetsfaktorn för NP. Med det hoppas jag kunna visa hur jag tänkt. Exemplet beskrivs i kod i modulen mw\_test\_file.py. Varje delmoment (med underliggande moment) kan köras i motsvarande modul.

1. Laddar två DataFilter (generellt + vinter), ett ToleranceFilter och RefValues.
2. Skapar DataHandler för rådata och läser in en txt-fil: Här behövs det göras en hel del jobb med att få in data på rätt sätt. Kanske ett av förstegen som vi pratade om.
3. Gör första filteringen med det generalla filtret (”first\_filter”). Får en ny DataHandler (”filtered\_data”) tillbaka.
4. Test med att spara och ladda ”filtered\_data”.
5. Skapar objekt för kvalitetsfaktorn (”qf\_NP”)
6. Adresserar data till kvalitetsfaktorobjektet (metod ”set\_data\_handler”):
   1. Om endast data\_handler ges adresseras denna ner till samtliga indikatorer. Om indikator ges adresseras data endast till den angivna indikatorn.
   2. I indikator-klassen adresseras DataHandlern till alla eller angiven underliggande parametrar/parameter.
   3. Nu har data distribuerats till alla eller angivna indikatorer/parametrar. Data ligger som DataHandlers i attributet self.data\_handler. Dessa är inga kopior utan pekar på den ursprungliga DataHandlern.
7. Filtrering på parameternivå. Detta sker enligt samma mönster som adressring av data i steg 6. Slutresultatet skapar en ny DataHandler under attributet self.data i varje angiven parameter. self.data är helt frånkopplad från den ursprungliga DataHandlern. Viktigt dock är att index är kvar för att kunna länka data i slutändan!   
   Typiskt kanske det generella filtret sätts på samtliga indikatorer varpå tex ett vinterfilter sätts på vinterindikatorerna.   
   All beräkningar kommer nu jobba mot self.data.
8. Beräkning av kvalitetsfaktor (viss begreppsförvirring råder här från min sida). Ingen sammanvägning har gjorts här. Här följer istället en beskrivning av hur matchande salt-data tas ut för beräkning av referensvärden.
   1. I metoden get\_ek\_value\_for\_par\_with\_salt\_ref i Indicator-klassen ges par, salt\_par, indicator\_name och tolerance\_filter.
   2. Parameterobjekt för aktuella parametern, salt-parametern och referensob
   3. En kopia av parameterobjektets column\_data görs (par\_df). Kopian kommer att fyllas på med beräknade värden med kringinformation.
   4. För varje index i par\_df görs en koll om värde för salt finns för samma index i salt\_df . Om värde hittas sparas saltvärdet och index (samma som det loopade indexet)
   5. Om värde inte hittas direkt (den snabba vägen) görs en sökning i parameterobjektet för salt. Raden för aktuellt index (innehåller all nödvändig metadata) samt toleransfilter ges till metoden ”get\_closest\_matching\_data”.
   6. I metoden görs först matchning mot de positioner, tider och djup som ryms inom den givna toleransen. Om matchning finns hittas först den närmaste positionen och därefter det närmsta djupet. Saltvärdet och tillhörande index returneras och sparas i df\_par.
   7. Referensvärde beräknas med hjälp av det funna saltvärdet.
   8. Gruppering (per station och sedan per år) och medelvärde beräknas.
   9. Minimum antal år kontrolleras från toleransfiltret.
   10. All information och data sparas i ett ClasificationResult-objekt för lättare hantering.

Jag har också gjort några olika diagram i draw.io för att visa flöden mm. Dessa ligger i mappen docs. Vet inte vilken typ av dokumentation vi tänker oss men för klassificering kanske det kan vara ide att göra sk. Sekvensdiagram. En bit på vägen för ett sådant skulle kunna se ut som ” sequence\_diagram\_quality\_factor\_NP.pdf”. Man får en bra överblick över hur och när olika klasser används.

**Frågor, funderingar och övrig status**

?: Ska get-metoderna i Parameter vara i DataHandler istället? Kan nog isf bara kopiera samtliga metoder samt kolla parameter. Hör parameter till col-data görs sökningen på ett sätt (det som nu är skrivet), är det row-data görs det på ett annat sätt.

?: TypeArea bör nog säkerställas för varje prov när data sorteras i ”första” DataHandler:n. Inget är idag gjort vad gäller TypeArea.

?: Loggning har jag inte jobbat med alls.

?: Hur (finns vissa metoder) och var data ska sparas har jag lämnat till Arnold.

?: Inget gjort vad gäller flaggor i data.

?: Om man vill filtrera bort data kanske man kan jobba med index.

?: Ingen koll finns på månader för RefValues.