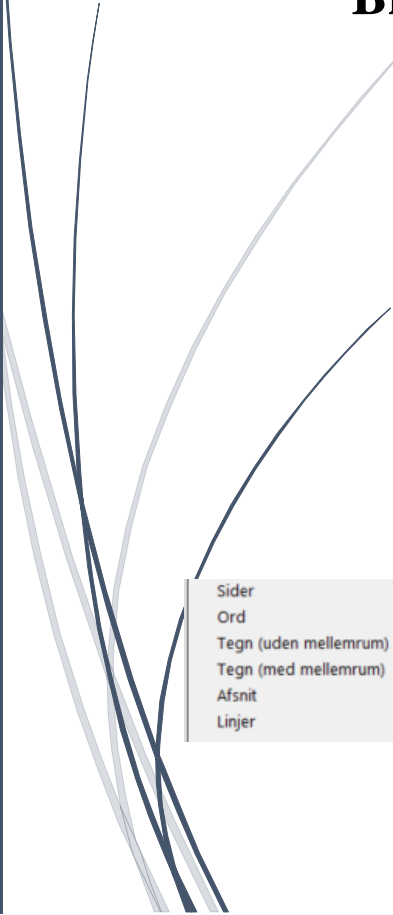




Brugervenlighed på Virk.dk



Sider	50
Ord	11.725
Tegn (uden mellemrum)	57.715
Tegn (med mellemrum)	70.802
Afsnit	731
Linjer	1.725

BACHELOR PROJEKT
Software Development
Mahnaz Karimi

Abstrakt

Denne rapport handler om hvordan jeg har analyseret dataene for at opnå en præsentation af brugeradfærd på indberetningsløsningerne på hjemmesiden Virk.dk ved hjælp af Sunburst Diagram og Histogram, for at visualisere hvordan brugerne på Virk.dk har bevæget sig, og hvor meget tid, der er brugt under hver URL og hvert felt. I rapporten begrundes de valg, jeg har foretaget i løbet af projektet. Imens jeg skrev denne rapport, har jeg styrket min viden om de teknologier, jeg har brugt. Dette projekt er lavet i samarbejde med Erhvervsstyrelsen som afslutning af min Softwareudviklings uddannelse på Copenhagen Business Academy i Lyngby.

En dataanalyse, hvor jeg skulle opnå en forståelse af brugeradfærd på specifikke løsninger under web-portalen Virk.dk. Portalen stilles til rådighed af flere styrelser til indberetning af person- og virksomhedsoplysninger (registrering af virksomheder, skatteoplysninger mm).

Hver enkelt løsning inden for Erhvervsstyrelsen har potentiale for at optimere brugernes indberetningsforløb. På sigt forventes dette realiseret igennem at minimere brugernes tid på indberetningerne (de forskellige "klik"), og gennem en præsentationen af indberetningerne på web interfacet.

I analysen indgår data fra indberetninger om opstart og lukning af virksomheder. Data er opsamlet siden starten af 2018. Data repræsenterer information fra hvert brugerklik på indberetningen. Den primære opgave er at forstå data og at klarlægge ovenstående problemstillinger. Ud over den direkte dataanalyse skal den opsamlede viden kobles med viden om, hvor der klikkes på hjemmesiden i indberetningen.

Jeg har fået to CSV-filer som kilde til dataanalyse af Erhvervsstyrelsen. Jeg har skrevet programmet til dataanalyse i Python. Jeg har analyseret data og oprettet nye CSV-filer. Disse blev brugt til visualisering af brugeradfærden. Jeg har også foretaget analyse af specifikke målgrupper.

For at analysere brugeradfærden har jeg undersøgt hvilke URL'er brugeren har besøgt. Ud fra dette fandt jeg URL-BrugerRejser, dvs. hvilken rækkefølge siderne er besøgt. Jeg har også undersøgt Indberetningsfelter på hvert URL. Disse kaldes Felt-BrugerRejser, hvilket vil sige rækkefølgen af felter der er blevet benyttet. Til sidst har jeg beregnet tidsforbrug under hvert URL og felt.

INDHOLDSFORTEGNELSE:

1) Resume	3
2) Virk.dk	3
3) Opgavebeskrivelse	4
4) Teknologier og værktøjer	5
4.1 Sunburst	7
4.2 Histogram og fraktil	8
5) Opgave	9
6) Database	10
7) Project design	12
7.1 Luk_Virksomhed mappe	13
8) Python Moduler	14
8.1 M-Url-Felt.py	15
8.1.1 Url-BrugerRejser	15
8.1.1.1 Rensning af URL	16
8.1.1.2 URL-BrugerRejser på Virk.dk	19
8.1.1.3 Erstatning tal i sted for URL i URL-BrugerRejser	23
8.1.1.4 Counter og gem data på CSV	24
8.1.2 Felt-BrugerRejser	25
8.1.2.1 feltnavn for Luk-Virksomhed	25
8.1.2.2 Feltnavn til Start-Virksomhed	27
8.1.2.3 Navngivning Standard for Feltnavn	27
8.1.2.4 Finde Felt-BrugerRejser	28
8.2 M-Time	32
8.2.1 Tids beregning for URL	32
8.2.2 Tidsberegning for felter	36
8.2.3 Gruppering af bruger	40
8.2.4 Oprettelse af histogram	41
9) Django	42

9.1. Installation af Virtualenv og Django	42
9.2 Django	43
9.2.1 Controller	43
9.2.2 Frontend (Views)	45
9.2.3 Django Database (Models)	46
10) Test	46
11) MySQL ind Docker	48
12) Programmets kørselstid	49
13) Konklusion	51
14) Yderligere referencer	52

1) Resume

Brugerne bruger for meget tid, når de indberetter eller sender en ansøgning på Virk.dk. Hjemmesidens indberetningsløsninger skal være optimale og brugervenlige da brugerens tid er værdifuld. Problemet kan heldigvis løses med dataanalyse. Sådan analyse kan bruges til andre undersøgelser af hjemmesider for at se hvor problematikken ligger.

2) Virk.dk

Når virksomheder skal kommunikere med danske myndigheder, f.eks. ved en indberetning eller en ansøgning, sker dette oftest via indberetningsløsninger på Virk.dk, som bestyres af Erhvervsstyrelsen. Mange af indberetningsløsningerne på Virk.dk bliver også udbudt af Erhvervsstyrelsen, bl.a. regnskabsindberetning og luk- og start virksomhed. Løsningerne er dog ikke altid så brugervenlige som de potentielt kunne være, og det vil Erhvervsstyrelsen gerne prøve at optimere på.

Erhvervsstyrelsen har mange indberetningsløsninger på Virk.dk, som ikke altid er lette for virksomheder at udfylde. Løsningen er at optimere løsningerne. Erhvervsstyrelsen vil hjælpe brugeren og gøre det lettere for burgeren som benytter løsning, da de gerne vil gøre det let at drive virksomhed. For at finde ud af hvad der kan optimeres, benytter afdelingen ML-Lab i

Erhvervsstyrelsen dataanalyse af opsamlet brugerdata. I dette projekt undersøges data opsamlet fra opstart og lukning af virksomheder.

Det problem er i første omgang undersøgt i kundecenteret, hvor de taler med brugere som har det svært. For dybere indsigt i tidsforbruget på hjemmeside og felter benyttes dataanalyse. Det er mere præcist og det rammer alle brugere, og også dem der ikke henvender sig til kundecenteret. Dataanalyse giver et bedre overblik over hele løsningen og dækker det hele, alle felter og undersider på hjemmesiden.

Det er interessant at løse problemet med programmering uden at man kender hjemmesiden rigtigt og felter under hver underside. Med dataanalysen fik jeg en oversigt af hjemmesider og deres felter under hvert URL og brugernes tidsforbrug på de enkelte undersider og hver felt tilhørende specifikke undersider.

Det er positivt at hjemmesiden er veloptimeret for både ejeren og hjemmesidens brugere. Det minimerer byrden på kundecenteret og Erhvervsstyrelsen kan spare ressourcer i kundecenteret, når brugeren har lettere ved at navigere rundt på løsningerne. Ud over det respekterer Erhvervsstyrelsen brugernes tid og vil gerne være et godt forbillede for brugervenlige indberetningsløsninger på Virk.de, så andre myndigheder kan finde inspiration i det og i sidste ende spare samfundet for besværlig administration.

Jeg har fundet potentiale for løsningsoptimering ved at bruge data analyse. En del af analysen var en visualisering af alle de bevægelser som brugerne foretager sig med Sunburst. Det er en letforståelig visualisering af data hvor man kan se rækkefølgen af brugerens handlinger på en flot måde.

3) Opgavebeskrivelse

Opgaven er arbejde med database som forgår i Linux computer i Jupiter i Python. Opgaven deles i nedenstående delopgaver:

- Først udarbejde på datasæt for Luk-Virksomhed
- Fjerner unødvendige detaljer i URL'er og gemme rensede URL'er i en ny kolonne
- Finder Unikke liste af URL'er!

- Sætter indeksettallet i sted for URL navne!
- Identificerer entydige URL-BrugerRejser!
- Ignorer gentagende URL adresser i URL-BrugerRejser hvis de er gentaget i rækkefølge!
- Visualiserer de afklarede URL-BrugerRejser med Sunburst!
- Finder nyt feltnavn udover kolonne 'feltid' eller 'feltnavn' og gemmes i ny kolonne.
- Finder Unikke liste af feltnavner!
- Sætter indeksettallet i sted for felt navner!
- Gemmes en liste af sekvens af indberetninger af hver bruger ind i en enhedsliste
- Finder de Felt-BrugerRejser under hver URL, men denne gang ignorer vi ikke de gentagende Felter adresser i Felt-BrugerRejser og visualiserer dem med Sunburst!
- Finder tidsforbrug på hver URL og visualisere dem histogram og fraktil!
- Finder tidsforbrug på hver felt, og visualiserer dem med histogram og fraktil!
- Deler data op i nogle grupper som i alt blev 14 og gentager alle de ovenstående analyser på de grupper
- Så skal køre analysen også for Start-Virksomhed
- Opretter moduler og en selvstændig applikation med Django

4) Teknologier og værktøjer

I denne del introducerer jeg de teknologier og værktøjer, jeg har brugt for at lave projektet:

- Linux
 - Jupyter
 - Python
 - Histogram(Fraktil)
 - Sunburst
 - Django
 - Docker
 - MySQL
 - Phpmyadmin

Linux:

Jeg fik en Linux maskine for at arbejde med under praktik. Kort sagt administrerer operativsystemet kommunikationen mellem din software og hardware. [2]

Jupyter:

Jeg har skrevet Python kode i Jupyter Notebook App, som er en server-klient program, der giver redigering og kører notebook dokumenter via en webbrowser. Anvendelser inkluderer: numerisk simulering, statistisk modellering, datavisualisering, maskinlæring og meget mere. [4]

Python:

Jeg brugte Python for skrive dataanalyse programmet i. Python fungerer på forskellige platforme (Windows, Mac, Linux). Python har en simpel syntaks, der ligner det engelske sprog. Python har syntaks, der giver udviklere mulighed for at skrive programmer med færre linjer end nogle andre programmeringssprog. Python kører på et tolkesystem, hvilket betyder, at kode kan udføres, så snart den er skrevet. Dette betyder, at prototyping kan være meget hurtig. [5]

Python tilbyder det alt hvad man har brug for nøjagtig, attraktiv, interaktive præsentationer og forståelig grafik som histogram som jeg har nyttet med for at visualisere data.

Histogram:

Et histogram er en type graf, der har brede anvendelser i statistik. Histogrammer give en visuel fortolkning af numeriske data ved at vælge antallet af datapunkter, der ligger inden for et scope af værdier. [6]

Jeg har brugt histogram for at visualisere tidsforbruget på hver felt og URL og antal af hver tidsforbrug!

Sunburst:

Et Sunburst-diagram bruges til at visualisere hierarkiske data, der er afbildet af koncentriske cirkler. Cirklen i midten repræsenterer rodnoden, hvor hierarkiet bevæger sig udad fra midten. Denne type visualisering viser hierarki gennem en række ringe, der er skåret for hver kategorien. Et Sunburst-diagram uden nogen hierarkiske data (et niveau af kategorier) ligner et donut-diagram. Imidlertid viser et Sunburst-diagram med flere niveauer af kategorier, hvordan de ydre ringe forholder sig til de indre ringe. [7]

Nedenfor er eksempler på Sunburst og histogrammer, jeg har arbejdet på og forklaret, hvordan de er brugt! Sunburst er i dette tilfælde baseret på JavaScript som kan kaldes af Python.

Django:

Django er en Python -baserede gratis og open-source web ramme, der følger den model-view-controller (MVC) arkitektoniske mønster og en let og uafhængig web-server til udvikling og test. Jeg har brugt Django for at lave en selvstændig applikation og valgt dette fordi Django understøtter Python.

Docker:

Docker er et open-source-projekt, som automatiserer ibrugtagning af computerprogrammer (med miljø) inden i softwarecontainere, ved at tilvejebringe et yderligere abstraktionslag og automation af virtualisering på styresystemniveau på Linux, OS X og Microsoft Windows. [11]

Jeg har brugt Docker for at importere Csv fil ind i MySQL

MySQL:

MySQL er en flertrådet SQL-databaseserver som understøtter mange samtidige brugere. I MySQL, der er en relationel database, kan man oprette tabeller. Forskellige browser-baserede programmer som phpMyAdmin og MySQL ABs egen Query Browser tilbyder en brugergrænseflade, der gør det nemmere at opdatere MySQL uden kendskab til SQL. [12]

Jeg har valgt MySQL fordi Csv-fil nemt kan importeres i dens tabeller.

phpMyAdmin:

phpMyAdmin er en browser-baseret samling af PHP scripts, der benyttes til at redigere MySQL databaser. phpMyAdmin hjælper med at danne de nødvendige SQL-kommandoer, som er de eneste "ordrer" databasen forstår, til manipulation af databasestruktur og -indhold. Det giver en bedre brugergrænseflade. [10]

Jeg har brugt for at vise data nemt på browseren.

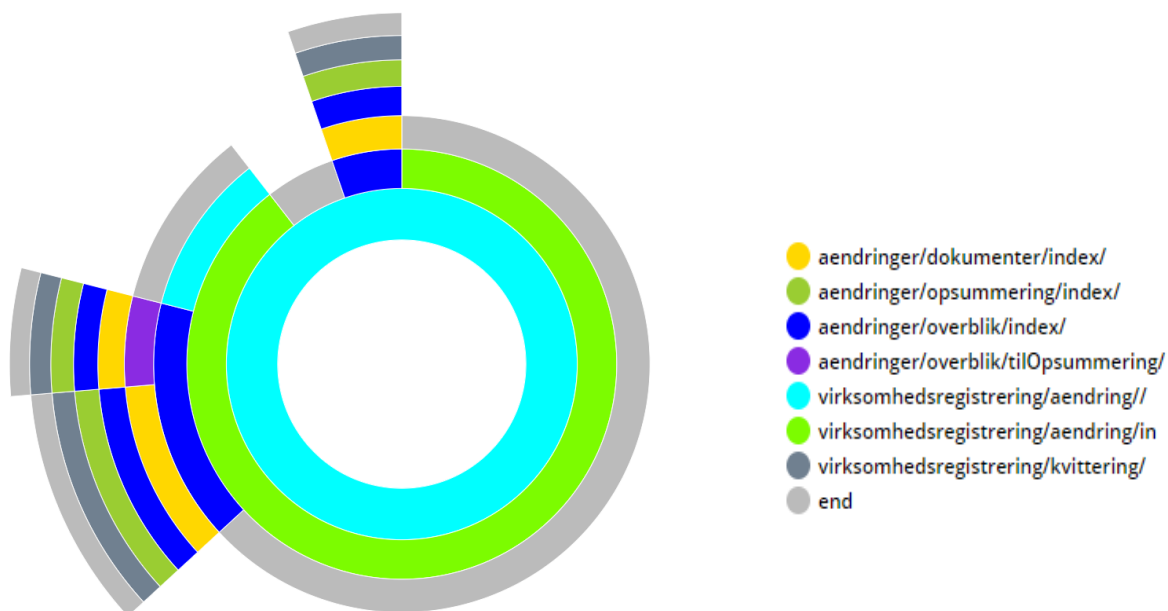
4.1 Sunburst

Vi har brugt Sunburst for at visualisere URL-BrugerRejser og Felt-BrugerRejser under hver underside af Virk.dk. Sunburst er JavaScript som bygges via Python. I den nedenstående billede kan I se en Sunburst som er dannet af en Csv-file som indeholder URL-BrugerRejser, men en Sunburst er ikke et billede og den har nogle funktioner som man kan benytte dem når du er i .html siden.

De .html side er blevet oprettet af en stykke kode som kan visualisere data og viser hierarki gennem en række ringe. Når vi er i .html side bevæger musen på hver ringe, så bliver de markere steder samme farver som står på billede og men andre steder hvor mus ikke peger på, mister deres farve. Så derfor hjælper med at fokusere på de rejser man vil kigge efter. På siden samtidig dukker op nogle titler i hierarki med en tekst streng tilhørende til den del som musen er på og viser hvad navnet er. Her er link til Sunburst

<https://bl.ocks.org/kerryrodden/7090426>

I den nedenstående Sunburst billede kan man se hvilken URL'er i række følge er blevet besøgt og hvor mange procent. Alt af de farve og masse giver mening for undersøgelsen!



Med det billede kan vi se at alle bruger er startet med URL:

● virksomhedsregistrering/aendring/

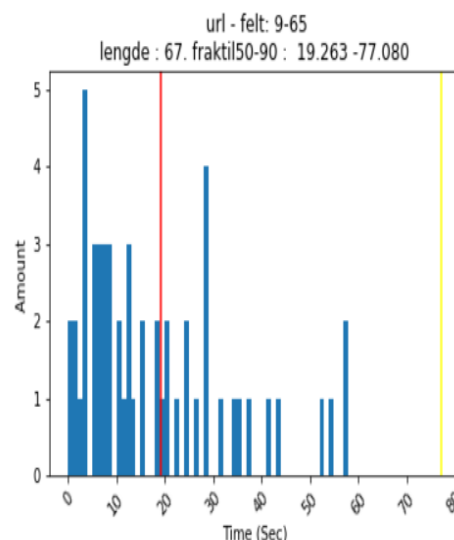
Næsten 80 procent er gået på Url:

● virksomhedsregistrering/aendring/in

...

4.2 Histogram og fraktil

Vi har lavet histogram og fraktil for tidsforbrug for alle felter og URL'er. Den histogram er for en visning tidsforbrug af et felt. I histogrammet på x-aksen er der tidsforbrug som beregnet på (Sec) og på y-aksen er antal af bruger trykket på felt. På histogram-billede har jeg skrevet en algoritme der vil printe de data som er stået over histogram- billeder. Der står tallet tilhørende til URL og tallet for feltet! Dvs. URL 9 og felt 65. Det fortælles også hvor mange data er analysen for som i det her er 67 klik af bruger. Der er også beregnet fraktil 50 og 90. for eks. fraktil 50 vil siges 50% af de antal bruger har brugt under end 19.263 sec og 90% er brugt tid under end 77.080 sec.



Billede Histogram (fraktil)

5) Opgave

Via data analyse skulle jeg opnå en forståelse af brugeradfærd på specifikke løsninger under web-portalen Virk. Portalen benyttes af flere styrelser til indberetning af person og virksomheds oplysninger (registrering af virksomhed, skatte oplysninger mm). Analysen af enkelt løsninger inden for Erhvervsstyrelsen skal give grundlaget for at optimere brugernes indberetningsforløb. På sigt forventes dette realiseret igennem at minimere brugernes tid på eneste indberetninger (de forskellige "klik"), og gennem en optimering af præsentationen af indberetningerne på web interfacet. Der var to forskellige datasæt som er blevet opsamlet og indhentet fra 2018. Gennem analyse af de data skal vi fik en statistik over hvilken undersider eller felt på Virk.dk har brugeren brugt meget tid!

Hver række af disse data er information om brugerens bevægelse på indberetningsløsningen. Data er repræsenterer information fra hvert bruger klik på indberetningen. Et sæt af data for de bruger har åbnet en virksomhed og et sæt af data for de bruger har lukket sin virksomhed. Hver gang en bruger bevæger sig på en løsning, så sendes data til Virks database, når en bruger klikke eller bevæger eller flytter musen ud af en felt, data bliver sent. Der var cirka 800,000 antal af data på Luk-Virksomhed og 7,600,000 antal af data på Start-Virksomhed. Så vil sige at inden et år var 800,000 klik på

undersider af Virk, da brugeren ville lukke virksomhed og 7,600,000 klik på de undersider hvor brugerne ville åbne virksomhed i et år! For at se koden fungerer korrekt og gøre hvad den skal gøre og for at have en bedre oversigt eller finde fejl hvis den dukker op, udarbejdede vi på færre data først. Efter fejl behandling køres dataene med flere data a gang.

Den primære opgave er at forestå data analysen rettet mod at klarlægge ovenstående problem stillinger. Ud over den direkte data analyse skal den opsamlede viden kobles med viden om hvor der klikkes på den relevante webside i indberetningen.

Information om web strukturens er leveret fra en anden gruppe i styrelsen, men de har ikke komplet information af selve Virk hjemmesiden. Informationer som URL adresser og feltnavne derfor gøre denne analyse besværligere!

6) Database

Dataene bliver gemt i Elasticsearch. Når man benytter Virk.dk logges de actions man tager. Der er et "tællerscript" som logger denne information og efter hvert "klik"/indtastning på Virk siden sender information til en lokal Virk server. Informationen indeholder information om bruger og action på web siden.

Data opsamles på modtager serveren til senere bearbejdning. Med passende tidsmellemløb sorteres den indkommende information fra de klik der er foretaget på virk portalen. De grupperes ift. til det forløb en specifik bruger har været igennem.

De grupperede data gemmes i Jason format i en Elasticsearch data base. Databasen indeholder alt logget informationer om virk brugerne over en længere årrække. Data kan hentes på to måder fra Elasticsearch. En generel information der er den samme for alle efterfølgende klik og en liste med information om hvert klik der høre til den rejse gennem virk brugeren har gennemført.

Den oprindelige information er delt i 20 databaser af sikkerhedsgrunde og skal derfor flettes sammen for at få det fulde information fra bruger rejsen inkl. følsom information om brugeren. Data strømmen fra virk gemmes til de Csv-filer.

Jeg fik de brugerdata opsamlet på to Csv-filer, en for dem har lukket virksomhed og en for dem har åbnet et virksomhed i 2018 som havde det samme struktur.

I nedenstående tabel står der de anvendte kolonner sammen med forklaringer på disse egenskaber, og i kan se i billede en oversigt over alle kolone er i databasen samt deres relationer.

Egenskaber	Beskrivelse
id	Bruger-ID
formURL	En underside af Virk.dk hvor brugeren har besøgt og indeholder brugerens informationer også!
RID	Resurser id
PID	Person id
feltId	Det felt navn hvor brugeren har bevæget musen på
feltNavn	Hvis der feltId var tom, så bruges dette til finde et navn for et felt
virkStartForloebId	Hver gang en bruger logger ind får en id.
serverTimeStamp	Det nøjagtige tidspunkt, hvor informationen blev sendt
handling	Informerer om musestatus. Hvis musen er på felten eller er komme ud af felten. Henholdsvis kaldes med to strenge: Focus eller blur!
cvrIndberet	Virksomhedsejerens id
Cvr	Person id
forloebVarighed	Person id
feltType	Person id

Der er blevet tilføjet to ekstra egenskab til hver datasæt. En der hedder 'clearURL' og 'clearField'. I starten skulle jeg rense URL'erne. Dette skyldes, at URL'erne indeholder brugerens detaljer.

Detaljerne viser hvem er logget ind på Virk-Hjemmesiden. Disse oplysninger gør det vanskeligt for os at finde unikke URL adresser og gøre også vanskelige når vi vil visualiser dem på histogram eller Sunburst, så skal de renses.

Derefter skal der findes et passende navn for hver felt. For at finde et navn kigges først i kolonne 'feltId', hvis den var tom kigges efter i den anden kolonne, 'feltNavn'. Felt navn skal være også












optimalt for den skal også bruges i visualisering! Efter det skal finde ud af hvor meget tid er brugerne brugt i hver underside og hvor meget tid brugen er brugt under hver felt. Så derfor er blevet to ekstra kolonne som indeholder beder navn for hver URL'er og felt.

Egenskaber	Beskrivelse
'clearURL'	'clearURL' er 'formURL' men den er blevet rensat af brugens detaljer.
'clearField'	datene under 'clearField' er 'feltId' men hvis 'feltId' er tom så bruges værdien af 'feltNavn' og hvis den var også tom, vil stå "NONE" og hvis der er en lang streng, vil den bliver forkortes !

Der er næsten fundet 28 unikke undersider 'clearURL' og 68 unikke feltnavne 'clearField' under datasæt Luk-Virksomhed og 128 unikke undersider og lidt over 1000 unikke feltnavne under datasæt Start-Virksomhed! Jeg vil forklare hver af disse sider separat nedenfor!

7) Project design

I den nedenstående billede er en skærm billede af en fil jeg har de Python sider og to mapper en til Luk-Virksomhed, som jeg vil gemme Csv-filer, Histogrammerne og Sunburst i. [Se billede B]

	.ipynb_checkpoints	04/09/2020 01.57	Filmappe
	__pycache__	04/09/2020 02.29	Filmappe
	EasyData	04/09/2020 01.57	Filmappe
	Luk_Virksomhed	04/09/2020 01.57	Filmappe
	tested_histogram	04/09/2020 01.57	Filmappe
	ConnectionToMysql.py	04/09/2020 01.57	JetBrains f
	M_PID_RID.py	24/08/2020 22.26	JetBrains f
	M_Time.py	04/09/2020 01.57	JetBrains f
	M_URL_Felt.py	04/09/2020 01.57	JetBrains f
	Main.ipynb	04/09/2020 01.57	IPYNB-fil
	sunburst_simple.ipynb	12/06/2020 00.46	IPYNB-fil

Billede B: En mappe, som vi har gemt Python-siderne i

M_URL_Felt.py	Her findes Virk undersider, URL-BrugerRejser og unikke liste af de URL-BrugerRejser og antal af dem og tilføjet Navngivning af felter og finde felt-BrugerRejser under hvilken url og unikke liste af de felt-BrugerRejser og antal af dem.
---------------	---

M_Time.py	En side til at finde tidsforbrug på hver URL og på hver felt tilhørende for hver URL
M_PID_RID.py	Alle bruger opdelt i 14 forskellige grupper. Alle ovenfor analyse gentages i de 14 grupper

7.1 Luk_Virksomhed mappe

For at jeg kunne gemme de Csv-filer for Sunburst og histogram billeder for Luk-Virksomhed der blevet oprettet to mapper. Som en fil som indeholder de forskellige filer har jeg oprettet for Star-Virksomhed også.

En fil som indeholder Histogram billeder og den anden fil indeholder Csv-filer for Sunburst. [Se billede B1]

Luk_Virksomhed		
Navn	Ændringsdato	Type
Histogram	05/06/2020 17.23	Filmappe
Sunburst	05/06/2020 17.09	Filmappe

Billede B1: de filer under Luk-Virksomhed

Under hver de ovenstående filer, er der en fil for alle bruger og en for 14 brugergrupper! [Se billede B2]

Luk_Virksomhed > Histogram		
Navn	Ændringsdato	Type
Alle_Bruger	05/06/2020 18.10	Filmappe
Gruppet_Bruger_PID&RID	05/06/2020 18.10	Filmappe

Billede B2: De filer under Histogram

Under Alle_Bruger har vi histogram for hver felt og Histogram for hver URL! [Se billede B3]

Luk_Virksomhed > Histogram > Alle_Bruger		
Navn	Ændringsdato	Type
Felter_Tidsforbrug_Histogram	05/06/2020 19.33	Filmappe
URL_Tidsforbrug_Histogram	05/06/2020 18.02	Filmappe

Billede B3: Filerne under Alle_Bruger

Under Histogram under Gruppet_Bruger_PID&RID har vi Histogram for at vise tidsforbrug for hver Felt og hver URL! [Se billede B4]

Luk_Virksomhed > Histogram > Gruppet_Bruger_PID&RID		
Navn	Ændringsdato	Type
Pid_Rid_Felter_Tidsforbrug_Histogram	05/06/2020 17.54	Filmappe
Pid_Rid_URL_Tidsforbrug_Histogram	05/06/2020 17.14	Filmappe

Billede B4: Filer under Histogram under Gruppet_Bruger

Under Sunburst under 'Gruppet_Bruger_PID&RID' har vi gemt Csv-filer ud af de Felt-BrugerRejser, URL-BrugerRejser og list af unikke felter tilhørende til hver URL! [Se billede B5]

Navn	Ændringsdato	Type	Størrelse
.ipynb_checkpoints	14/08/2020 10.59	Filmappe	
Felt_Brugerresje_Count_Csv	22/08/2020 18.50	Filmappe	
Unikke_URL_Feild_Count	31/08/2020 16.56	Filmappe	
URL_Brugerrejser_Count_Csv	22/08/2020 18.50	Filmappe	

Billede B5: Csv-filer for URL-BrugerRejser og Felt-BrugerRejser

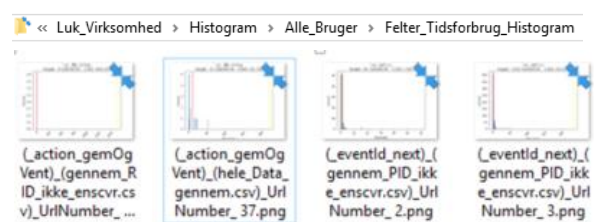
Så er der 14 forskellige Csv-filer under for hver af gruppe af data som indeholder URL-BrugerRejser! [Se billede B6]

Navn	Ændringsdato
gennem_PID_enscvt.csv	10/04/2020 08.59
gennem_PID_ikke_enscvt.csv	10/04/2020 08.59
gennem_RID_enscvt.csv	10/04/2020 08.59
gennem_RID_ikke_enscvt.csv	10/04/2020 08.59
gennem_RID_PID_enscvt.csv	10/04/2020 08.59
gennem_RID_PID_ikke_enscvt.csv	10/04/2020 08.59
hele_Data_gennem.csv	10/04/2020 08.59
hele_Data_ikke_gennem.csv	10/04/2020 08.59
ikke_gennem_PID_enscvt.csv	10/04/2020 08.59
ikke_gennem_PID_ikke_enscvt.csv	10/04/2020 08.59
ikke_gennem_RID_enscvt.csv	10/04/2020 08.59
ikke_gennem_RID_ikke_enscvt.csv	10/04/2020 08.59
ikke_gennem_RID_PID_enscvt.csv	10/04/2020 08.59
ikke_gennem_RID_PID_ikke_enscvt.csv	10/04/2020 08.59

Billede B6: Csv filer af URL-BrugerRejser for forskellige bruger

Histogram-billeder for Felter bliver gemt med et navn som er feltnavn. Det feltnavn som Histogram er blevet oprettet for, og tallet efter felt-navnet viser hvilken URL nr. tilhører feltet til!

[Se billede B7]



Billede B7: Histogrammer for felter under en URL

Under Sunburst under Alle_Bruger bliver gemt Csv-filer, en Csv fil for unikke Felt-BrugerRejser samt antal af dem, og en for unikke URL-BrugerRejser og antal af dem! [Se billede B8]

Navn	Ændringsdato
Unique_FeltRejser_&Count.csv	15/04/2020 20.26
Unique_UrlRejser_&Count.csv	15/04/2020 20.26

Billede B8: Csv-filer af unikke URL og felt BrugerRejser

8) Python Moduler

Moduler er Dot py-filer, der består af Python-kode. Enhver Python-fil kan refereres til som et modul. Moduler kan definere funktioner, klasser og variabler, som man kan henvise til i andre Python .py-filer eller via Python-kommandolinjetolken. I Python får man adgang til moduler ved hjælp af import erklæringer. [3]

De nedenstående moduler er alle de moduler som er importeret fra andre biblioteker for dette analyse!

```
import pandas as pd
import numpy as np
import os
import time
from collections import Counter
import datetime
from datetime import timedelta
import time
import matplotlib.pyplot as plt
import random
```

Billede for importeret modul

8.1 M-Url-Felt.py

Hvis vi vil tale lidt enkle, kan vi sige, at mange URL'er bestemmer placeringen af en fil et sted i webområdet for os. En fil, der kan være et foto, en film, en simpel tekst, et webom formateringsfil (CSS), et program eller noget andet. URL er kendt som en web-adresse, en URL (Uniform Resource Locator) er en form for URI og en standardiseret navnekonvention til adressering af dokumenter, der er tilgængelige via Internettet. Vi ser normalt http- og https-kommunikationsprotokoller, der bruges til at få adgang til websteder; Der er dog andre protokoller (såsom FTP, telnet). Det sidste afsnit af en URL indeholder den adresse og de endelige detaljer, der fører os til destinationen (eller fil eller kilde). [9]

8.1.1 Url-BrugerRejser

Her vil vi finde hvordan brugerne har bevæget sig på de forskellige web-adresser af Virk undersider. Derfor finder vi URL-BrugerRejser for at finde hvilken vej brugerne burde tage og hvor mange gang er blevet taget forkert og hvor mange er taget korrekte. Hvis der er mange som er fortaget fejl så må det være at de ikke navigeres godt på de sider, Det føres til at brugeren bruger meget tid. Så derfor er at vigtig at finde ud af hvordan de har bevæget sig!

Når vi finder URL-BrugerRejser, giver det ikke mening når to ens URL adresse står ved siden hinanden for det betyder at en bruger har vært på samme URL adresse. Vi vil finde de bevægelser når hver bruger fortager en ændring dvs. at skifte siderne på virk. Derfor for at finde URL-BrugerRejser sletter vi de gentaget URL'er hvor de står ved siden af hinanden i en list. Vi sletter

ikke de ganget feltnavne som er gentaget i rækkefølge, når vi finder felt-BrugerRejser. Fordi de gentagelser vil betyde at en felt er ikke nemt at udfyldes og brugeren bruger meget tid på dette felt.

8.1.1.1 Rensning af URL

For at start analyse så kalder vi Data med at bruge pandas bibliotek.

```
Data= pd.read_csv('Luk_virksomhed.csv',sep=';',dtype='str',nrows=1000)
```

Billede 1.a: Csv-fil. Separat med ";"! Type af data skal strengen! kalder kun 1000 de første rækker. Det opbevares i et Dataframe kaldet 'data'

Vi giver sti hvor Csv ligger og informere hvordan data bliver separeret og hvilke type data de er og hvor mange række af data vi vil have!

I det her skal vi finde unikke URL'er og derefter finde ud af hvordan brugeren har bevægede sig på undersiderne på Virk hjemmeside. For at finde URL-BrugerRejser kigger vi efter kolonnen 'formURL'. 'formURL' indeholder adressen af undersider af Virk.dk som brugeren er besøgt. De URL adresser indeholder brugerens detaljer som ikke skal bruges til dataanalysen. For at analyse data i 'formURL' kaster vi data af denne kolonne på en separate Dataframe Kaldte 'df'. [Se billede1.c]

```
df=Data['formURL']
```

Billede 1.c: Opretter en Dataframe ud af en kolonne!

Vi gøre dette for at have en bedre fokus på URL adresser og finde ud af hvordan vi kan rense de unødvendig detaljer. De logins detaljer som er i URL'er har forskellige strukturer. En af de strukturer kan ses i den nedenstående eksempel:

<https://erst.virk.dk/virksomhedsregistrering/aendring/index?forlarId=11111&forlbId=1da6a1-a111-1a11-881a-u777777>

Ovenfor er en eksempel på den URL som skulle renses! Efter undersøgelse, indsås i midt af nogle URL'er er der nogle detaljer som startede med "? ". Efter spørgsmålstegn er der noget bogstaver, tegn og tal som ikke kan bruges og skulle slettes.

Med kun en linje kode vil hver URL splittes med spørgsmålstegn og derefter lægges i en dataframe med to kolonne som vil indeholde de streng før og efter spørgsmålstegn! [Se billede2]

```
df = Data.fromURL.str.split("?", expand = True, n=6).rename(columns={0:"Befor", 1: "Efter"})
```

Billede2: Opretter en tabel med to kolonner. Ændrer Dataframe kolonnes navne til "BEFOR" & "EFTER" split-data med "?" giver navn for columns samtidig!

Man kan se hvad den stykke kode gør på tabellen som vist nedenfor:

	Befor	Efter
1	https://erst.virk.dk/virksomhedsregistrering/aendring/	None
2	https://erst.virk.dk/virksomhedsregistrering/aendring/index	formularId=11111&forloebId=idfaf6a1-a111-1a11-881a-u777777

Efter splitting med "?" sletter vi strengen efter "?" med drop metode. [Se billede3]

```
df = df.drop(['Efter'], axis=1)
```

Billede3: Fjernes kolonne med navn 'efter', gem det selv igen.

URL kan splittes med "/" så splitter vi det. [Se billede4]

```
df = df.Befor.str.split('/', expand = True, n=10)
```

Billede4: Splitter 'Befor' kolonne med '/' og gemmes igen!

Med at printe 'df' vil man se data som er splittet med "/"! [Se billede5]

	0	1	2	3	4	5	6
0	https:	erst.virk.dk	virksomhedsregistrering	aendring			None
1	https:	erst.virk.dk	virksomhedsregistrering	aendring	index		None
2	https:	erst.virk.dk	virksomhedsregistrering	aendring	index		None
14	https:	erst.virk.dk	aendringer	X00-AA-00-XX	overblik		index
15	https:	erst.virk.dk	aendringer	X00-AA-00-XX	opsummering		index
16	https:	erst.virk.dk	aendringer	X00-AA-00-XX	opsummering		index

Billede5: Alle strenger mellem backslash er separeret

Så fylder vi de non pladser med et tom streng for at vi ikke får fejl senere. [Se billede6]

```
df=df.fillna("")
```

Billede6: med Fillna. Fylder vi tomme pladser med tom streng!

I billede 5, under kolonne nr. 4, er der nogen streng der ligner at være brugerens detalje igen og skal slettes for det bør ikke være en del af URL-adressen. Dette streng er bestået af tal og bogstaver som er adskilt med en linje. Når i en for loop vil vi samle datene igen med skråstreg, vil vi ignorerer den del som ligner brugerens detalje. Så samler URL'erne igen med den nedenstående kode. [Se billede7]

```
1 ll = []
2 for i in np.arange(len(df)): # df er dataframen med alle data fra formURL,renset efter ? og derefter er blevet splittet med "/"
3     line= df.iloc[i]# tager hver række af data, i hver celle af rækken er de del mellem '/'som vi har fået fra de splittet url
4     http=""
5     if line[0].startswith('http'):
6         for j in np.arange(len(line)): # Loop med længde af den i eneste splittet url med (/)
7             lineSplit= line[j]
8             if lineSplit is None:# hvis den del af splittet url er tomt, så gøres ingen ting
9                 http=http
10            elif len(lineSplit)> 0:
11                t2=lineSplit.split("-") # split elementen with(-) for vi ved at der findes noget i urlen der ligner kode og har
12                if (len(t2)==1):
13                    try:
14                        int(lineSplit) # hvis splitet del ikke kan converts to int kommer i exept del og bliver added
15                    except:
16                        http=http+lineSplit+"/"
17
18 else:|
19     http="NONE"
20
21 ll.append(http) # http er rene url hvor det bliver added til en list
```

Billede7: Der er to for-loop i hinanden. Første loop-størrelse er list af alle de url adresser, næste loop er størrelse af selve URL adressen!

I den nedenstående tabel er alle forklaring om den ovenstående kode I billede nr.7

1	ll en tom List	Vi introducerer en tom liste, som vi vil gemme de samlet URL
2	Nr.1 For-Loop	Størrelse af det loop er størrelse af datafremme kolonne 'formURL'. Så er den samme længde af de oprindelig data
3	Hver linje	Hiver ud hver gang en linje af tabellen ud. Tabellen i billede nr. 5
4	Tom streng 'http'	Tom streng vil sætte hver eneste URL sammen med en skråstreg ud fra de splittede data, og danner en URL adresse igen
5	Nr.1 if-statement	I starten tjekkes hvis første del af URL starter med 'http'. En URL'er starter enten med http eller https, ellers den er ikke en URL-Adresse og derefter i Else-statement 'http' vil blive til "NONE".
6	Nr.2 For-Loop	Størrelse af den loop er længde af hver række. Hver række er URL splittet med skråstreg!

7	lineSplit	Så hives ud henholdsvis hver del af data ud fra hver række og gemmes i variablen.
8	Nr.2 if-statement	Tjekke hvis der er en tom streng så vil program ikke gøre noget!
9	Nr.1 elif-statement	Hvis den ikke er en tom streng tjekker programmet om længde af strengen er større end null!
10	t2. tom st	Hvis længde af strengen er større end null, vil strengen splittes med '-'
11	Nr.3 if-statement	Hvis den længde af splittet data er et, så er det ikke som en brugers id <code>X00-AA-00-XX</code> Efter analyse opdagede jeg at der også er et lang tal i mellem URL'er som ikke skal være en del af URL adressen og skal slettes
12	Try	Tjekker hvis den del kan konverteres til tal, så vil programmet ikke tilføje den del i URL adressen!
13	Except	Når objektet er nået hertil, vil det betyde at den ikke har '-' i mellem og ikke er et tal, så vil den sættes sammen med en skråstreg og gemmes i en objekt som vil danne URL!
14	Append metode	Så i det sidst addes den samlet URL adresse i en liste. Listen er liste af rensede URL'er

Efter at vi fyldte listen med rensede URL adresser så tilføjer vi denne liste under ny kolonne kaldte "clearURL". [Se Billede 8]

```
Data.insert(13, '{}'.format(columnName), ll, True) # make a new column data and put list in it
```

Billede8: Den nemmeste måde at oprette en ny kolonne og udfylde den med data i ll liste med de rensede URL'er.

Med den nedenstående kode får vi en kortere URL adresse. [Se Billede 9]

```
ll=[w.replace("https://erst.virk.dk/", "") for w in ll]
ll=[w.replace("#0", "") for w in ll]
ll=[w.replace("https://erst.virk.dk/", "") for w in ll]
ll=[w.replace("#", "") for w in ll]
```

Billede9: vi kan rydde op lidt mere på URL navner!

8.1.1.2 URL-BrugerRejser på Virk.dk

For at se hvordan hver enkelte brugeren bevægede sig rundt på hjemmesiden, finder vi liste af unikke bruger-id. Derefter baseret på hver brugers id hiver vi data ud og så samler deres URL'er

```
uniqIdUserAndClearURL = pd.DataFrame(Data.groupby(['id']).agg({'clearURL': '#Mellemrum# '.join}))
```

som de har besøgt og samler sammen med en streng #Mellemrum# in i mellem og gemmer dem i en ny Dataframe. [Se Billede 10]

Billede 10: Finder et unikt bruger-id og samler alle deres URL sammen med en streng i mellem, som jeg har valgt at strengen #MELLEMRUM#

Her dannes to kolonne, en med unikke id nummer, og foran hver unikke id står der en lang streng som samlet alle en brugers URL adresse fra 'clear URL' og er lagt sammen med en streng #Mellemrum#! Nedenstående billede viser hvad den stykke kode gøre! [Se Billede 11]

	id	clearURL
0	X11x1x- x0x0-11ss 7-1xx1-00	virksomhedsregistrering/aendring/ #Mellemrum# virksomhedsregistrering/aendring/index/ #Mellemrum# virksomhedsregistrering/aendring/index/ #Mellemrum# virksomhedsregistrering/aendring/index/ ...
1	X21x1x-0-11ss7-1 xx1-0000500x1x0	aendringer/overblik/index/ #Mellemrum# aendringer/dokumenter/index/ #Mellemrum# aendringer/dokumenter/index/ #Mellemrum# aendringer/dokumenter/index/ #Mellemrum# aendringer/dokumenter/inde...

Billede 11: 1 første kolonne, hver unik brugers id! næste kolonne er en lang streng, samlet alle URL'er sammen for hver bruger

Her skal vi slette alle de URL'er som er gentaget mere end en gang efter hinanden. Fordi det ikke er vigtigt, hvor meget bruger forbliver på en side og her ønsker vi bare at blive klar over rækkefølgen af de anvendte side og tiden som brugeren har brugt på hver underside! Vi gør det også fordi som nævnt tidligere vi har ikke haft Frontend dokumentation og ønsker at nå en vision om det hvordan brugere på undersider af Virk indberetningsløsninger er rejst rundt som de sandsynligvis er vejledt via selve hjemmesiden! I det nedenstående kode slettes de gentagne URL'er! [Se Billede 12]

```
def deleteRepeatedUrl(uniqueUserGruppeClearUrl, columnname):
    import numpy as np
    lenUniqueUserGruppeClearUrl=len(uniqueUserGruppeClearUrl)
    listOfNotRepeatetURL=[]
    for i in np.arange(lenUniqueUserGruppeClearUrl): #uniqIdGruppeClearUrl er en list af uniq id og de trykk de har taget
        ClearHttp=[]
        gruppetClearUrl= uniqueUserGruppeClearUrl['{}'.format(columnname)][i] #split de url som vi lagt sammen med Mellemrum
        gruppetClearUrlSplitet=gruppetClearUrl.split('#Mellemrum#')
        nloop=len(gruppetClearUrlSplitet)-1 #fordi i metoden bruge en element frem
        for j in np.arange(nloop):
            url1 = gruppetClearUrlSplitet[j].strip()
            url2= gruppetClearUrlSplitet[j+1].strip()
            if j != (nloop-1):
                if url1!= url2:
                    ClearHttp.append(url1)
            else:
                if url1!= url2:
                    ClearHttp.append(url1)
                    ClearHttp.append(url2) #renser de url hvis der findes noget gentagelse efter hinanden
            else:
                ClearHttp.append(url2) #renser de url hvis der findes noget gentagelse efter hinanden
        listOfNotRepeatetURL.append(ClearHttp)
    uniqueUserGruppeClearUrl["notRepeatetClearURL"]=listOfNotRepeatetURL
    return uniqueUserGruppeClearUrl
```

BILLEDE 12 Her sletter vi de u repeterede URL'er!

1	En tom List listOfNotRepeatetURL	Vi initialisere en tom liste, som vi ønsker at gemme alle de ikke-Gentaget URL-BrugerRejser i!
2	Nr.1 For-Loop	Størrelse af det loop er længden af unik list af unikke bruger Id som vi har samlet deres URL'er
3	En tom List ClearURL	Vil gemme de streng ikke-Gentaget URL for hver eneste bruger i! Den bliver en tom liste i For-Loop nr. 1.
4	gruppetClearUrl	Her hives ud henholdsvis hver række af data der indeholder en lang streng af de URL'er for hver bruger
5	En Streng gruppetClearUrlSplitet	Her splittes hver streng som er samlet alle brugerens 'clearURL' med #mellemrum#
6	Et tal nloop	Størrelse af Dataframe er antal af de URL'er er gemt i en streng med #mellemrum# imellem minus en. Minus en fordi vi i det næste loop vil vi kalde en URL frem af. Vi gøre det for at undgå får fejl i programmet, når programmet når til det sidste række
7	Nr.2 For-Loop	Størrelse af den loop er antal af URL'er gemt for hver bruger med #mellemrum# minus en
8	En Streng url1	Loopes i URL'er, og hives URL'er af gang Med strip() metode kan vi fjerne tom plads omkring hver URL
9	En Streng url2	Den næste URL i Loopen og samlinger to af dem af gang!
10	Nr.1 If-Statement	Hvis vi ikke er i det sidste række. Minus en igen fordi variabel i Loopen bliver aldrig lige med størrelsen af For-Loop, fordi den starter fra nul af.
11	Nr.1 If-Statement in Nr.1 If-Statement	Hvis vi ikke er i den sidste række og hvis url1 og url2 ikke er ens, tilføjes kun den første URL adresse, og hvis URL'er er ens så program gøre ingen ting! For bedre at forstå hvad programmet gør, brugte nummeret i sted for URL adresser for at se hvordan programmet skal rydde op i listen sådan så ingen af de lignende objekter står ved siden af hinanden. For. Eks. den liste {[1,1,1], [2,6,6], [2,5,5,2]} bliver til den liste {[1], [2,6], [2,5,2]} Koden kan bruges til andre undersøgelser der ønsker at rydde op i en liste og ikke ønsker nogen objekter, der ligner hinanden står ved siden af hinanden!
13	Nr.1 If-Statement in Nr.1 Else-Statement	Hvis vi er I det sidste loop og de URL'er ikke er ens, så vil begge URL'er bliver tilføjet til listen.
14	Nr.1 Else-Statement in Nr.1 Else-Statement	Hvis vi er I det sidste loop og de URL'er er ens, så vil en af de URL'er bliver tilføjet!
15	Liste istOfNotRepeatetURL	Tilføjes den streng med de ikke-gentaget URL'er på listen

Så vil lægges den list i Datagrammet under ny kolonne ” newClearURL”. [Se Billede 13]

```
uniqidUserAndClearURL["newClearURL"] = listOfNotRepeatetURL
```

Billede 13: En ny kolonne i Dataframe, som vil samlede hver unikke brugers URL-adresser

Så vil den nedenstående tabel udskrives når vi kalder ’uniqidUserAndClearURL ’. [Se Billede 14]

id	clearURL	newClearURL
0- 7- 4- 32	virksomhedsregistrering/aendring/ #Mellemrum# virksomhedsregistrering/aendring/index/ #Mellemrum# virksomhedsregistrering/aendring/index/ #Mellemrum# virksomhedsregistrering/aendring/index/ ...	[virksomhedsregistrering/aendring/ virksomhedsregistrering/aendring/index/, aendringer/overblik/index/ aendringer/dokumenter/index/, aendringer/overblik/index/ aendringer/overblik/tilOpsummering/]
5- 4- 32	virksomhedsregistrering/aendring/ #Mellemrum# virksomhedsregistrering/aendring/index/ #Mellemrum# virksomhedsregistrering/aendring/index/	[virksomhedsregistrering/aendring/ virksomhedsregistrering/aendring/index/]

Billede 14: Vi har brugeridéer og deres URL'er og en kolonne med gemte de ikke-repeterede url'er;

Med Counter finder vi ud af de unikke URL-BrugerRejser og antal af dem. [Billede 15]

```
dict1 = Counter([tuple(i) for i in uniqIdGruppeClearUrl["newClearURL"]])  
UrlRejserAndCount = pd.DataFrame(data = {'ListofUrlRejser': list(dict1.keys()),  
                                         'countUrlRejser': list(dict1.values())})
```

Billede 15: Med Counter, kan man finde unikke lists af en list og antal af dem!

Den ovenstående Counter vil printes på den nedenstående billede! [Se Billede 16]

	urlList	countUrl
0	(virksomhedsregistrering/aendring/, virksomhedsregistrering/aendring/index/, aendringer/overblik/index/, aendringer/dokumenter/index/, aendringer/overblik/index/, aendringer/overblik/tilOpsummering/)	2
1	(virksomhedsregistrering/aendring/,)	26

Billede 16: Vi kan se at unikke URL-BrugerRejser og antal af dem

For at sætte tal i sted for en URL finder vi først en unik liste over URL'er. [Se Billede 18]

```
listOfuniqUrl=pd.DataFrame(Data['clearURL'].unique()).rename(columns={0:"urlName"})  
listOfuniqUrl['urlNumber'] = list(np.arange(len(listOfuniqUrl)))
```

Billede 18: første linje opretter en unik liste og giver samtidig et navn til kolonne, der indeholder unik url-adresse. I næste linje bruger vi et liste af tal ud af vores data længder.

Første linje laver en unik liste af ’clearURL’ og giver samtidig et navn på kolonne. I linje to, bruger vi antallet af URL'er til at oprette en liste med range af længden. Så alle URL'er har et tal foren sig.

8.1.1.3 Erstatning tal i sted for URL i URL-BrugerRejser

Den Unikke list af URL'er vil ses ud. [Se Billede 19]

	urlName	urlNumber
0	virksomhedsregistrering/aendring/	0
1	virksomhedsregistrering/aendring/index/	1

Billede 19: Den liste af Unik URL navner og nummer tilhørende til hver URL

Ved at bruge den nedenstående stykke kode, har jeg lavet en Dictionary af en list. Dictionary ved hjælp af Keys() og Value() kan finde en værdi meget hurtigere og den kommer ud af søgning hver gang den har fundet en værdi som programmet søger efter. Så derfor vil den nedsætter tids kørsel. I nedenstående billede kan i se hvor nemt kan man lave en Dictionary i Python! [Se Billede 20]

```
HashlistOfUrl= listOfuniquUrl.to_dict()
```

Billede 20: en nemt mode for at oprette en hash list\Dictionary af en Dataframe!

Så printer vi den Dictionary ud!

```
HashlistOfUrl
```

```
{'urlName': {0: 'virksomhedsregistrering/aendring/',  
1: 'virksomhedsregistrering/aendring/index/',  
6: 'virksomhedsregistrering/kvittering/'},  
'urlNumber': {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6}}
```

Billede 21: En Dictionary af unikke URL'er og deres tilhørende nummer!

For at have en bedre og mere specifik visning af URL-BrugerRejser så vil vi erstatter det svarende tal til hver URL. Med det stykke kode ned under erstatter vi tal i stedet for URL. [Billede 22]

```
1 numberInstedOfUrl=uniqIdUserAndClearURL["newClearURL"].to_frame()  
2 for i in np.arange(len(numberInstedOfUrl)):  
3     newClearUrlRow=numberInstedOfUrl.newClearURL[i]# hives ud hver urlrejse  
4     for j in np.arange(len(newClearUrlRow)): # hives ud hver url  
5         for x in np.arange(len(listOfuniquUrl)):  
6             if (newClearUrlRow[j]==HashlistOfUrl['urlName'][x]): # findes tal der passer til den url gennem unik url  
7                 newClearUrlRow[j]=HashlistOfUrl['urlNumber'][x]
```

Billede 22: Vi erstatter nummer i sted for URL Adresse.

1	En Dataframe numberInstedOfUrl	Vi gemmer de ikke-gentaget URL'er i den Dataframe som kan også kaldes URL BrugerRejser. I kan se i billede nr. 13 og 14.
2	Nr.1 For-Loop	Størrelsen af det loop er størrelsen af alle de ikke-gentaget URL'er.
3	En Dataframe newClearUrlRow	I den Dataframe har vi kun en række af data. Kun en URL-BrugerRejser af gang som vi får det i Loopen.
4	Nr.2 For-Loop	Størrelsen af loop er antallet af de URL'er som er i hver URL-BrugerRejse
5	Nr.3 For-Loop	Størrelse af den loop er størrelse af unikke list af URL'er!
6	Nr.1 If-Statement	Den tjekker hvis den URL som vi får ud af den nuværende URL-BrugerRejser er lige med en af de værdier stående i Dictionary af URL'er.
7	Erstatning af tal	Hvis den betingelse i nr. 6 bliver opfyldt Dvs. har fundet tallet til URL, så vil tallet bliver erstattet i sted for URL i den del af koden!

Så printer vi list af URL-BrugerRejser som har tal i stedet for navn! [Se Billede 23]

```
numberInstedOfUrl
```

	newClearURL
0	[0, 1, 2, 4, 2, 5]
1	[0]
2	[0, 1]

Billede 23: En liste af URL-BrugerRejser

Vi vil i fremtiden bruge disse numre i stedet for URL'er. Dette vil forhindre at vi får en omfangsrig tabel så derfor får vi en bedre overblik på tabellen og også vil fremskynder søgningen!

8.1.1.4 Counter og gem data på CSV

Indtil her har fundet ud af URL-BrugerRejser og antal af hver og derefter gemmer vi de unikke URL BrugerRejser og antal af hver rejser i en Csv fil til senere brug når vi vil visualisere URL-BrugerRejser ind i Sunburst. Vi gemmer også unikke URL'er og deres tal i en Csv, for at have dem klar til at brug senere [Se Billede 24]

```
listOfuniqUrl[['urlName', 'urlNumber']].to_csv("Luk_Virkksomhed/UniqueUrl.csv", sep=";", index=False)
```

```
UrlRejserAndCount[['countUrl', 'urlList']].to_csv("Luk_Virkksomhed/UniqueUrlRejserAndCount.csv", sep=";", index=False)
```

Billede 24: Oprettelse af Csv som separates hver linje med "; ". Når vi har to kolonner kan vi bruge [[]] for at give som en argument.

Jeg fandt URL-BrugerRejser indtil her, næste analyse er at finde Felt-BrugerRejser!

8.1.2 Felt-BrugerRejser

Erhvervsstyrelse har ikke rigtig oversigt over feltnavn på hver URL under Virk.dk. Nogle felter er blevet navngivet på Virks hjemmeside gennem brugerens valg eller er baseret på hvad de har indtastet på feltet. Dette gør analysen svært, fordi blandt de feltnavn er nogle navne som består af ulæselig eller lang tekst. Det var ikke så besværligt at finde feltnavn i Luk-Virksomhed datasættet men det var svært at finde feltnavn i Start-Virksomhed datasættet og der var behov for en masse datarens. For at finde et navn for et felt navn kigget efter 'feltId' og 'feltNavn' og det valgte navn af de to vil lægges i ny kolumne kaldes 'clearField'.

Der var mange navne på felter som vi fik fejl når vi ville visualisere dem på Histogram eller Sunburst. De navn ligesom 'Begrundelse for godkendelse\xa0*'. Som vi med at printe feltnavne før det at fejlen sker, kan man se hvad grundet er at program fejler. De tegn i det eksempel efter backslash kunne man ikke se i Dataframe, den kunne kun ses når man printes en Dictionary ud. Jeg vil nedenfor forklare, hvordan vi kunne finde feltnavne på datasæt relateret til Luk-Virksomhed, og derefter vil jeg forklare hvordan vi har optimeret feltnavn på Start-Virksomhed.

8.1.2.1 feltnavn for Luk-Virksomhed

Efter analyse på feltnavne eller feltid på Luk-Virksomhed indså vi at der er nogle af lange feltnavne på flere sprog. Her vil du se to eksempel af de lange navn:

Dansk	Polsk
'\n\n Jeg erklærer at betalingserklæringen:\n\n\tIndeholder virksomhedens CVR-nummer, navn og adresse\n\t\tAt kapitalejer/kapitalejere har erklæret, at al gæld, forfalden som uforfalden, er betalt, og at virksomheden er besluttet opløst\n\t\tAt erklæringen er underskrevet af samtlige kapitalejere.\n\t\tAt erklæringen er dateret\n\t\tAt samtlige kapitalejere er angivet med navne, adresse og underskrift\n\t\tAt samtlige kapitalejere er gjort bekendt med, at hvis de afgiver urigtige oplysninger, kan dfocus',	'\n\n Oświadczam, że deklaracja płatności:\n\n\tZawiera firmowy numer CVR, nazwę i adres\n\t\tWłaściciele / właściciele kapitału zadeklarowali, że wszystkie zadłużenie jako nierozliczone jest wypłacane, a spółka została rozwiązana\n\t\tŻe deklaracja została podpisana przez wszystkich akcjonariuszy.\n\t\tTo, że deklaracja jest datowana\n\t\tWszyscy właściciele kapitału otrzymują imiona, adresy i podpisy\n\t\tŻe wszyscy udziałowcy są świadomi, że jeśli dać im fałszywe informblur',

Derefter har vi forkortet de lange navne som i kan se i den nedenstående metode. [Se billede 25]

```

kanFocus="kan dfocus"
kandbluer="kan dblur"
theyMayBfocus="they may bfocus"
theyMayBblur="they may bblur"
theyFocus="they focus"
theyBlur="they blur"
im_informblur="im falszywe informblur"
im_informfocus="im falszywe informfocus"

1 def reduce_field_name(rowclear,feildlist):
2     if(kanFocus in rowclear):
        feildlist.append("betalingserklæringen")
        return feildlist
    elif(kandbluer in rowclear):
        feildlist.append("betalingserklæringen")
        return feildlist
    elif(theyMayBfocus in rowclear):
        feildlist.append("declaration payment may")
        return feildlist
    elif(theyMayBblur in rowclear):
        feildlist.append("declaration payment may")
        return feildlist
    elif(theyFocus in rowclear):
        feildlist.append("declaration payment they")
        return feildlist
    elif(theyBlur in rowclear):
        feildlist.append("declaration payment they")
        return feildlist
    elif(im_informblur in rowclear):
        feildlist.append("declaration payment'falszywe inform'")
        return feildlist
    elif(im_informfocus in rowclear):
        feildlist.append("declaration payment'falszywe inform'")
        return feildlist
3     else:
4         feildlist.append(rowclear)
        return feildlist

```

1. Den metode tager vi to variabler en som er feltnavnet og en liste af feltnavne.

2. Hvis den feltnavn indeholder dette streng så vil tilføjes en bestemt streng til listen.

3. Hvis ingen af de betingelser var ikke opfyldt så vil den selve feltnavn tilføjes til listen

4. Så vil det kun returnere den list af feltnavn!

Billede 25: Denne metode tager to variabler, som er feltnavnet og en liste over feltnavn. I sidste ende, uanset om navnet er forkortet eller ej, vil det blive tilføjet til denne liste

Den ovenstående metode er brugt til at finde et feltnavn i den næste analyse. [Se Billede 26]

```

1 gatherFieldidOrFieldName=[]
2 for i in np.arange(len(Data)):
3     feltid = Data.feltId[i]
4     if (type(feltid)!=type(0.0)):
        feltNavn =Data.feltNavn[i]
5         if (type(feltNavn)!=type(0.0)):
            gatherFieldidOrFieldName.append("NONE")
6         else:
            gatherFieldidOrFieldName=reduce_field_name(feltNavn,gatherFieldidOrFieldName)
7     else:
8         gatherFieldidOrFieldName=reduce_field_name(feltid,gatherFieldidOrFieldName)
Data.insert(13, '{}'.format(columnName), 11, True) # make a new column data and put list ind i

```

Billede 26: Her finder vi et nyt navn på hver felt og sætter dem på listen i rækkefølge og tilføjer listen til hoved dataen i slutningen.

1	gatherFieldidOrFieldName	En tom liste der vil samle de uændrede eller de ændrede feltnavne
2	Nr.1 For-Loop	Størrelse af den loop er størrelse af Main data!
3-	Feltid	I rækkefølge i hver række af data kigger vi først efter 'feltId'
4	Nr.1 If-Statement	Den tjekker hvis feltId er Nan, så vil kigge efter 'feltNavn'
5	Nr.1 If-Statement I Nr.1 If-Statement	Den tjekker hvis feltNavn er tom, så vil tilføje en streng "NONE" til listen
6	Nr.1 Else-Statement in Nr.1 If-Statement	Hvis feltNavn ikke er tom så vil feltNavn sendes til ovenstående metode til at analysere
7	Nr.1 Else-Statement	Hvis feltId i starten ikke er tom vil sendes til ovenstående metoden for at analysere

8	Inserting af data	Til sidst efter For-Loop er færdig så vil den liste, som har nye felt navne, indsættes i den ny kolonne med navn "ClearField"
---	-------------------	---

8.1.2.2 Feltnavn til Start-Virksomhed

Vi analyserede feltnavn i Start-Virksomhed er lidt anderledes end Luk-Virksomhed. For der var mange unikke feltnavne derfor vi ikke kunne navngive som jeg har gjort i ovenstående metode for Luk-Virksomhed.

For at opnå et optimale feltNavn for begge datasæt ser vi først på 'feltId' og hvis den var tom, programmet vil kigge efter 'feltNavn'. Hvis den også er tomt, så tilføjes en streng 'NONE' til listen og til sidst vil listen placeres under 'clearField' i det hoved data.

Efter at have oprettet en unik list af den nye kolonne 'clearField' var der næsten 5000 feltnavn. Så lige efter at kigge på unik liste over feltnavne fandt vi ud af, at der er et antal lange navne blandt i feltnavne, som øger programmets udførelsestid og gør det vanskeligt at visualisere dem.

I denne gang hvis i de feltnavne er mere end 15 mellemrum vil erstattes med en streng 'NONE'. Derefter faldt antal af unikke feltnavn til 1300. Derefter indså vi også blandt disse unikke navne er der mange feltnavne var en streng lignende: '00000000-xxx5-11y9-927x-0000569y0d02' som ligner brugerens ID. Når der er som denne format blandet i datene erstatter vi en streng 'LikeUSERID'. Derefter antallet af unikke feltnavne faldt ned til 1000. [Se billede 27]

```
def clearFelt(Data,columnName1,columnName2,newColumnName):
    Data=pd.DataFrame(Data)
    gatherFieldIdOrFieldName=[]
    for i in np.arange(len(Data)):
        feltId = Data['{}'].format(columnName1))[i]
        if (type(feltId)!=type(0.0)): # hvis felt id ikke var None eller float type :
            if((len(feltId.split(" "))>15)):
                gatherFieldIdOrFieldName.append("NONE")# hvis den er feltid er for Langt så vil addes "NONE"
            elif("\n" in feltId):
                feltId=feltId.replace("\n","")
                gatherFieldIdOrFieldName.append(str(feltId))
            elif(len(feltId.split("-"))>2):
                gatherFieldIdOrFieldName.append('LikeUSERID')
            else:
                gatherFieldIdOrFieldName.append(str(feltId))
        else:
            feltNavn =Data['{}'].format(columnName2)[i]
            if (type(feltNavn)!=type(0.0) or (len(feltNavn.split(" "))>15)):
                gatherFieldIdOrFieldName.append("NONE")
            else:
                if("\n" in feltNavn):
                    feltNavn=feltNavn.replace("\n","")#vil fjerne hvis de er en nylinjetejn '\n'
                    feltNavn=feltNavn.strip()
                    gatherFieldIdOrFieldName.append(str(feltNavn))
                elif(len(feltNavn.split("-"))>2):
                    gatherFieldIdOrFieldName.append('LikeUSERID')
                else:
                    gatherFieldIdOrFieldName.append(str(feltNavn))
    Data.insert(14,'{}'.format(newColumnName),gatherFieldIdOrFieldName, True)
    return Data
```

1. For-Loops størrelsen er lig med main datas størrelse.
2. Hvis feltid ikke var tom.
3. Så Hvis feltnavn var mere end 16 ord, så erstattes 'NONE'.
4. Hvis der var '\n' i feltid så erstatter vi en tom streng i stedet.
5. Hvis der mere en to bindestreg i så erstattes 'LikeUSERID'.
6. Der var ingen af de ovenstående betingelser, så feltnavn addes.
7. men hvis feltId var tom, og hvis feltNavn var også tom eller var mere end 16 ord i feltNavn så erstattes 'NONE'
8. Til sidst sættes den list, vi har udarbejdet, som den nye kolonne 'clearField'

Billede 27: Finer et feltnavn, opret en ny kolonne, og sæt fundet feltnavn i

8.1.2.3 Navngivning Standard for Feltnavn

Til sidst fandt jeg en god metode som normaliserer feltnavne og opretter et standard navn. De standard navne bruges til at vise histogram-billeder og også for at navngive når Csv eller histogram billeder vil gemmes. [Se billede 27b]

```
def format_filename(s):
    import string
    """Take a string and return a valid filename constructed from the string.
    Uses a whitelist approach: any characters not present in valid_chars are
    removed. Also spaces are replaced with underscores.

    Note: this method may produce invalid filenames such as '', `.` or `..`
    When I use this method I prepend a date string like '2009_01_15_19_46_32_'
    and append a file extension like '.txt', so I avoid the potential of using
    an invalid filename.

    """
    valid_chars = "-_.() %s%s" % (string.ascii_letters, string.digits)
    filename = ''.join(c for c in s if c in valid_chars)
    filename = filename.replace(' ', '_') # I don't like spaces in filenames.
    return filename
```

Billede 27b: fjerner alle de tegn som ikke skal være blandet i mellem et navn for en fil.

Her slettes tegn eller mellemrum og danner et standard navn der vil hjælpe at fejl ikke dukker op i undersøgelse!

8.1.2.4 Finde Felt-BrugerRejser

Under hver URL vil vi gerne finde på hvilken rækkefølge hver brugeren har bevæget sig over. For at finde Felt-BrugerRejser, har vi ikke brug for de data som under kolonnen 'handling' står strengen 'blur'. Fordi disse data viser at brugeren har flyttet musen ud fra feltet som brugeren i forvejen var ind. For at finde felt-BrugerRejser kan kun kigge efter de felter som er blevet besøgt og ikke når de blev forladt. Så derfor med denne stykke kode nedenfor sletter vi de data hvor 'handling' er 'blur'.

[Se Billede 28]

```
1  ilistHandlingHaveBlur=[]
   for i in np.arange(len(Data)):
       handling = str(Data.handling[i])
       if (handling=='blur'):
2      ilistHandlingHaveBlur.append(i)
   Data.drop(Data.index[ilistHandlingHaveBlur], inplace=True)
3  Data.reset_index(drop=True, inplace = True)
4
```

Billede 28: Fjerner de data som deres handling er 'blur'

1. For-Loop størrelse af hele data
2. Henholdsvis hvis i en række af data under kolonne 'handling' står der 'blur' så den indeks af række gemmes i en liste.
3. Så slettes de indeks vi har samlet i en list.
4. Så vil indekset bliver reset så får vi en indeks med rækkefølge igen.

Antal af rækker af data faldes, så kørselstiden reduceres!

Vi skal finde felt-BrugerRejser men denne gang skal vi ikke slette de gentaget felter efter hinanden.

For at finde ud af hvordan de bruger har benyttet felterne på hver URL, kigger programmet på to række data ad gang i rækkefølge. Hvis der er ens bruger og på en Url, samler alle felter på en liste som i et forløb er efter hinanden.

Hvis de to felter er ikke under ens url eller ikke er besøgt af ens bruger så vil URL-id til den første række data gemmes i listen og feltnavne for første række data gemmes i den anden listen og derefter listen bliver tom for det næste forløb!

Det nedenstående kode er oprettet for at lave dette algoritme. [Se Billede 29]

```
def findeFeildOrServerTime(Data, listOfUniqUrl, HashlistOfUrl, columnName):
    listOfUrlid=[]
    List=[]
    listOfList=[]

    aNumberLessOfDataLength= (len(Data)-1) # hvor vi i metoden kalder vi en value frem af, for at undgå få exeption
    toNumberLessOfDataLength= aNumberLessOfDataLength-1 # når i en for Loop ikke vi at i rammer sidste element

    for i in np.arange(aNumberLessOfDataLength): #Loop for at finde hver linje i Data

        row1=Data.iloc[i]
        row2=Data.iloc[i+1]

        nummerUniqUrl= findeUrlNum(listOfUniqUrl, HashlistOfUrl, row1) # beregner tal for en url

        if (i!=(toNumberLessOfDataLength)): # når vi er en til sidste data row
            if((row1.id==row2.id) and (row1.clearURL.strip()==row2.clearURL.strip())):# når en brugeren er ens
                #og brugeren er på en url.
                List.append(row1.get(key= columnName))# så tager vi den første rækkes serverTimeStamp og sætter
                #i en list.

            else:# Hvis brugeren eller urlen ikke er ens så addes den første række til servertime list og bruge
                List.append(row1.get(key= columnName))# en method for at adde de servertime list og url id
                listOfList, listOfUrlid=appendMethod(List, nummerUniqUrl, listOfList, listOfUrlid)
                List=[] #tømmes servertime list når brugern eller url er ikke ens

        else: # når er den sidste række af data
            if((row1.id==row2.id) and (row1.clearURL.strip()==row2.clearURL.strip())):# hvis brugeren og url er ens
                List.append(row1.get(key= columnName))# addes begge serverTimeStamp og url id addes til
                List.append(row2.get(key= columnName))
                listOfList, listOfUrlid=appendMethod(List, nummerUniqUrl, listOfList, listOfUrlid)
            else:# hvis brugern eller url ikke er ens i den sidste 2 række
                List.append(row1.get(key= columnName))#Først addes den første række serverTimeStamp og dens url
                listOfList, listOfUrlid=appendMethod(List, nummerUniqUrl, listOfList, listOfUrlid)
                List=[] # så tømmes time liste
                List.append(row2.get(key= columnName))# addes næste serverTimeStamp
                nummerUniqUrl= findeUrlNum(listOfUniqUrl, HashlistOfUrl, row2)# finde url nr og kalder den 'appendMethod'
                listOfList, listOfUrlid=appendMethod(List, nummerUniqUrl, listOfList, listOfUrlid)

    listOfUrlidAndFeltRejse = pd.DataFrame(data ={'Urlid': list(listOfUrlid),
                                                'List': list(listOfList)})
    return listOfUrlidAndFeltRejse
```

Billede 29:

Finding Felt-BrugerRejser.

1	Tre Tomme lister	Efterfølgende: -En tom liste for at samle op de unikke URL id. De URL som er besøgt fra brugeren i rækkefølge baseret på tid. -En tom liste for at samle op de felter som de i rækkefølge er under enes URL og brugt af en bruger. -En liste for at samle de liste af felter af gang!
2	To tal	Et tal som er størrelsen af Main data minus en, for vi i Loopen kalder en række data frem af.

		Et tal som vil vise at vi er i det sidste loop, der er to mindre af størrelsen af Main data, for 'i' i Loopen bliver aldrig lige med størrelse af selve Loopen for den starter med 0.
3	Nr.1 For-Loop	Størrelse af den loop er størrelse af Main data minus en! for vi kalder to række data frem i Loopen
4	Et tal numberUniqUrl	Her programmet vil finde tallet tilhørende for URL adresse for den første række af data!
5	Nr.1 If-Statement in If-Statement	Den tjekker hvis vi ikke er i den sidste række og så tjekkes at brugers Id og URL adressen er ens, så vil tilføjes den første feltnavn til listen indtil brugen id og URL er ens. I stedet for timeList kunne bruges list og i stedet for listOfTimeList kunne bruges listToList Vi behandler sidste række anderledes, for vi skal også behandle data nummer to feltnavn!
6	Nr.1 else-Statement in Nr. 1 If-Statement	Hvis vi ikke er i det sidste række og hvis to række data er ikke ens eller brugeren ikke er under ens URL så tilføjes den først feltnavn, og derefter vil den list tilføjes til stor enhedsliste og tilføjes også den tal tilhørende til URL id til listen 'listOfUrlid', og derefter den 'feltList' bliver lige med tom liste i bliver klare til næste loop. Senere fortælles hvad 'appendMethod' er!
7	Nr. 1 If-Statement 1Else statement	Hvis vi er i den sidste loop, hvis der er ens bruger i hver række og ens URL adresse så vil feltnavn på begge rækker gemmes i listen, og en af Url nummer bliver også tilføjet til listen med appendMethod.
8	Nr.1 else-Statement in Nr. 1 else-statement	Når vi er i det sidste række og hvis enten brugerens id eller Url adresse ikke var ens, så vil den første feltnavn gemmes i listen og derefter gemmes Url nr. til den anden List 'listOfUserid' og så vil felt-listen tilføjes til 'listOfFieldList' og derefter vil felt list bliver tom. Så vil vi gentage dette til række nr. to, først addes til felt list og så felt list tilføjes til en list, derefter gemmes række nr. to Url-nummer til Listen 'listOfUserid'.
9	listOfUrlidAndFeltRejse	Her sættes to liste sammen da de har ens længde. Så opretter vi en Dataframe med to kolonne. En liste af URL id og en liste af felt-liste tilhørende til hver URL <i>som er brugt af hver ene bruger</i> Så i det sidste returneres Dataframe! Nedenfor med at printe vil du se hvordan den Dataframe vil ses ud i billede nr. 32!

For at gøre koden læseligere har jeg oprettet de nedenstående to metoder og brugt dem in i programmet. [Se Billede 30]

I nedenstående metode med fire variabler, en felt list som gemmes under en list der indeholder felt-liste og en URL nr. som skal tilføjes i URL liste. Derefter vil returnere to lister, en liste af felt-liste og en liste med URL nummer! [Se Billede 30]

```
def appendMethod(feildList,nummberUniqUrl,listOfFeildList,listOfUrlid):
    listOfFeildList.append(feildList)
    listOfUrlid.append(nummberUniqUrl)
    return listOfFeildList,listOfUrlid;
```

Billede 30: adder en liste in I anden liste og samtidig samler URL nummer i en anden list!

In nedenstående metode med to input en URL navn og en list af unikke URL'er. Den metode finder tallet tilhørende til URL navn ud fra i den unikke liste og returnere det! [Se Billede 31]

```
def findeUrlNum(listOfUniqUrlEfterDeleteBlur,urlRow):|
    for k in np.arange(len(listOfUniqUrlEfterDeleteBlur)):# finde tal nummer til uniq url
        if ((urlRow.clearURL.strip())==(listOfUniqUrlEfterDeleteBlur.urlName[k].strip())):
            nummerUniqUrl= HashlistOfUrlEfterDeleteBlur['urlNumber'][k]
    return nummerUniqUrl
```

Billede 31: vil finde nummeret korresponderende til url-adressen

Her kan ses hver Felt-BrugerRejser og tilhørende URL-id foran sig. Data er sorteret i rækkefølge som serveren har modtaget data. [Se Billede 32]

Urlid	FeildList
0 0	[NONE]
1 1	[NONE, cvnummer]
2 0	[NONE]
3 1	[NONE, _action_soeg]

Billede 32: Liste over URL'er og lister over felter under dette URL-nummer

I stedet for navne på felter erstatter vi tal lig nøjagtig som vi gjorde for URL'er tidligere. Efter erstatte tal i sted for felt navn tilføjes liste med tal i en liste under ny kolumne.

```
def enters_Numbers_Instead_Of_Felt_Name(listOfUrlIdAndFeltRejse,HashListfeild):
    listOfAllNumOFFeildList=[]
    for i in np.arange(len(listOfUrlIdAndFeltRejse)):# List of feltList sammen med urlider
        listFieldRow=listOfUrlIdAndFeltRejse['FeildList'][i] #row fra felt lister
        listOfFeilt=[]
        for j in np.arange(len(listFieldRow)): # Looper i hver row, hver row har måske list af urler
            field=listFieldRow[j] # første element
            if (type(field) == list): #hvis første element er også en liste, har vi bruge for kun den første element
                field = field[0]
            for x in np.arange(len(HashListfeild["FeildName"])): #her finder vi tal nummer til hver felter
                if field==HashListfeild["FeildName"][x] or ((type(field)==type(0.0)) & (type(HashListfeild["FeildName"][x])!=type(0.0))):
                    # Non element giver besværligheder for at undgå det checker vi det at element er non(Float) eller ej
                    listOfFeilt.append(x)
                break
            listOfAllNumOFFeildList.append(listOfFeilt)

    listOfUrlIdAndFeltRejse["listOfNumberOFFelt"]=listOfAllNumOFFeildList
    return listOfUrlIdAndFeltRejse
```

Så vil man se Felt-BrugerRejser som er indtastet tal i sted for feltnavn:

Urlid	FeildList	listOfNumberOFFelt
0 0	[NONE]	[0]
1 1	[NONE, cvnummer]	[0, 1]
2 0	[NONE]	[0]
3 1	[NONE, _action_soeg]	[0, 2]

Billede 34: en liste af URL'er og deres feltbrugerrejser med navne og også tal

Den nedenstående kode opretter Csv-filer i en loop med antal af Loopen størrelse. Csv-filer vil indeholde all Felt-BrugerRejser til hver URL. [Se Billede 34]

```
def make_Csv_For_FeldList_for_every_URL(listOfUniqUrl,listOfUrlIdAndFeltRejse, path):

    try:
        for i in np.arange(len(listOfUniqUrl)):# I in en størrelse af unike url liste efter delete data med ''
            csvname= path+'FeltBrugerRejser_Under_URL{}.csv'.format(i)
            # opretter
            #dynamic navn for csv file som vil indeholde felt brugerrejser og antal af dem

            listQuery=listOfUrlIdAndFeltRejse.query('Urlid==@i')[['listOfNumberOfFelt']]#hiver data ud fra
            # hver unike url id og bruger dens felterrejser
            countfeltamount = Counter([tuple(i) for i in listQuery['listOfNumberOfFelt']])# beregner antal af de uniq felter

            FeltRejserAndCount = pd.DataFrame(data ={'feltlis': [list(t) for t in countfeltamount.keys()],
                                                    'count': list(countfeltamount.values())}) # opretter en DF med unike felt brugerrejse og antal af dem

            FeltRejserAndCount.to_csv(csvname,sep=";", index=False)# kaster den DF to csv med den dynamic navn fra ovnpå
        print("CSV file saved!")
    except Exception as e:
        print(str(e))
```

Billede 34: Opretter Csv-filer af feltrejser for hver undersider

1	1 For-Loop	Starter en loop Størrelse af den loop er lig med unikke URL listen.
2	En streng Csvname	Et dynamik navn der vil indeholde en sti som Csv-filer vil gemmes. De dynamisk navn for de Csv-filer der vil indeholde strengen "Felt-BrugerRejser_Uner_URL" og i det ende af navnet vil erstattes de dynamik URL nr.
3	En QUERY listQuery	Med Query metode vi kan kalde henholdsvis de data hvor URL_id er lige med den tal i loopet og derefter hentes alle de feltlister tilhørende til den URL_id.
4	Et Counter listQuery	Der oprettes Counter på en list som indeholder feltlister sammen med Url. [se billede 34]
5	Oprettelse af Csv-filer	Så vil programmet gemme de unikke felt-BrugerRejser tilhørende på hver URL sammen med antal af de unikke felt-BrugerRejser i Csv-filer som har et dynamisk navn. Navnet ender med tilhørende URL-id

Så skal vi finde tidsforbrug til hver URL og felt, som jeg vil forklare i det næste afsnit!

8.2 M-Time

8.2.1 Tids beregning for URL

For at finde hvor meget tid er brugt under hver URL, bruger vi den algoritme som vi har brugt for felt-BrugerRejser, men denne gang i stedet for 'clearField' bruger vi kolonnen 'ServerTimeStamp'.

[Se Billede 35]

```
def findeFeildOrServerTime(Data, listOfuniqUrl, HashlistOfUrl, columnsName):

    listOfUrlid=[]
    List=[]
    listOfList=[]

    aNumberLessOfDataLength= (len(Data)-1) # hvor vi i metoden kalder vi en value frem af, for at undgå få exeption
    toNumberLessOfDataLength= aNumberLessOfDataLength-1 # når i en for loop ikke vi at i rammer sidste element

    for i in np.arange(aNumberLessOfDataLength): #Loop for at finde hver linje i Data

        row1=Data.iloc[i]
        row2=Data.iloc[i+1]

        nummberUniqUrl= findeUrlNum(listOfuniqUrl,HashlistOfUrl,row1)# beregner tal for en url

        if (i!=(toNumberLessOfDataLength)): # når vi er er en til sidste data row
            if((row1.id==row2.id) and (row1.clearURL.strip()==row2.clearURL.strip())):# når en brugeren er ens
                #og brugeren er på en url.
                List.append(row1.get(key= columnsName))# så tager vi den første rækkes servertimeStamp og sætter
                #i en list.

            else:# Hvis brugeren eller urlen ikke er ens så addes den første række til servertime list og bruge
                List.append(row1.get(key= columnsName))# en method for at adde de servertime list og url id
                listOfList,listOfUrlid=appendMethod(List,nummberUniqUrl,listOfList,listOfUrlid)
                List=[] #tømmes servertime list når brugern eller url er ikke ens

        else: # når er den sidste række af data
            if((row1.id==row2.id) and (row1.clearURL.strip()==row2.clearURL.strip())):# hvis brugeren og url er ens
                List.append(row1.get(key= columnsName))# addes begge serverTimeStamp og url id addes til
                List.append(row2.get(key= columnsName))
                listOfList,listOfUrlid=appendMethod(List,nummberUniqUrl,listOfList,listOfUrlid)
            else:# hvis brugern eller url ikke er ens i den sidste 2 række
                List.append(row1.get(key= columnsName))#Først addes den første række serverTimeStamp og dens url
                listOfList,listOfUrlid=appendMethod(List,nummberUniqUrl,listOfList,listOfUrlid)
                List=[] # så tømmes time liste
                List.append(row2.get(key= columnsName))# addes næste serverTimeStamp
                nummberUniqUrl= findeUrlNum(listOfuniqUrl,HashlistOfUrl,row2)# finde url nr og kalder den 'appendMethod'
                listOfList,listOfUrlid=appendMethod(List,nummberUniqUrl,listOfList,listOfUrlid)

    listOfUrlIdAndFeltRejse = pd.DataFrame(data ={'UrlId': list(listOfUrlid),
                                                'List': list(listOfList)})
    return listOfUrlIdAndFeltRejse
```

Billede 35: Oprettelse af en Dataframe med to kolonne, en til de liste af tider og en list over de URL nr. som brugeren har været på når de benyttet felt

I den nedenstående tabel kan man se hvilke data den ovenstående kode returnerer. [Se Billede 36]

	listOfUrlid	listOfServerTime Stamp
0	0	[2018-01-01T00:37:04.923]
1	1	[2018-01-01T00:37:30.644, 2018-01-01T00:37:38.753, 2018-01-01T00:38:00.913]
2	0	[2018-01-01T13:53:59.989]

Billede 36: En liste af URL nr. og en liste af de tider som brugerne har været på virk.dk

Her for at se hvor meget tid er brugt på hver Url gennem en bruger af gangen, kigger vi på den første tid og sidste tid og dividerer dem med hinanden. Med det nedenstående kode kan vi finde ud af hvor meget tid der er brugt på hver URL. [Se Billede 37]

```

1      datetimeFormat = '%Y-%m-%dT%H:%M:%S.%f'
2      ListOfDifferenceTid=[]

3      for i in np.arange(len(listOfurlIdAndServerTimeStamp)):
4          listOfDate=listOfurlIdAndServerTimeStamp['listOfServerTimeStamp'][i]
5          if ((len(listOfDate))==1):
              ListOfDifferenceTid.append("0")#hvis der er kun en i list så adder vi 0
              #hvis len af data >1, tager vi den sidste og første række serverTimeStamp og beregner tid forsked mellem dem.

          else:
              date1=datetime.datetime.strptime(listOfDate[0], datetimeFormat)#første række
              date2=datetime.datetime.strptime(listOfDate[len(listOfDate)-1], datetimeFormat)#sidste række
              diff = abs(date2-date1)
              ListOfDifferenceTid.append(str(diff.total_seconds()))

```

Billede 37: beregning af tid på Url

1	datetimeFormat	Det er tidformat under 'ServerTimeStamp'. Vi vil bruge den når vi vil oversætte dato til tid. Det er for tidsberegninger!
2	En tom list	Tom liste for at samle de beregnede tider op.
3	For-Loop	Længde af loop er længde af lister med alle kunders tider. [Se Billede 36]
4	If-Statement	Hvis der kun er én tid på listen, betyder det at brugeren kun har været en gang på url, så tilføjer vi 0 til listen!
5	Else-statement	Hvis der er flere tider på listen, vil programmet kun oversætte dato til tiden for den første og den sidste på listen, så beregnes tids forskel på sekunder og programmet tilføjer tidsforskellen på listen.

Til sidst vil programmet lægge sammen listen med de beregnet tider ud af time-list og liste af URL'er ind i en Dataframe! [Se Billede 37]

```

listOfurlIdAndDifferenceTimeStamp= pd.DataFrame(data ={'listOfUrlid': list(listOfUrlid),
              'listOfDifferenceTimeStamp': list(ListOfDifferenceTid)})

```

	listOfUrlid	listOfDifference Time Stamp
0	0	0
1	1	30.269
2	0	0
3	1	7.297

Billede 37: liste af URL og beregnede tidsforbrug

Derefter samler vi alle tider baseret på Url-id og lægger sammen i en Dataframe! [Se Billede 38]

```

collectTime = listOfurlIdAndDifferenceTimeStamp.groupby(['listOfUrlid']).agg({'listOfDifferenceTimeStamp': ' #Mellelrum# ' .join
collectTime = collectTime.reset_index() # når vi reset index så vi får også en komumns med den listOfUrlid

```

Billede38:en Dataframe med url-id og de tids forbruger

I det nedenstående billede ses hvordan programmet samler data! [Se Billede 39]

	listOfUrlid	listOfDifferenceTimeStamp
0	0	0 #Mellemrum# 0 #Mellemrum# 0 #Mellemrum# 4.929 #Mellemrum# 828.516 #Mellemrum# 0 #Mellemrum# 19.365 #Mellemrum# 0 #Mellemrum# 0 #Mellemrum# 0 #Mellemrum# 24.29...
1	1	30.269 #Mellemrum# 7.297 #Mellemrum# 0 #Mellemrum# 0 #Mellemrum# 74.451 #Mellemrum# 13.048 #Mellemrum# 0 #Mellemrum# 0 #Mellemrum# 0 #Mellemrum# 6.608 #Mellemrum# 11.933 #Mell...
2	2	0
3	3	56.009

Billede 39: samler alle tider på hver url sammen med en streng!

Med den nedenstående kode folder vi de tider ud og lægger dem sammen! [Se Billede 40]

```
listofSecoun=[]
listoflistSecoun=[]
listOfTotalSec=[]
lenOfCollection=len(collectTime)# længde af en dataframe gruppet by url id samler tider

collectTime=pd.DataFrame(collectTime)
for i in np.arange(lenOfCollection): # Len of url ider
1     totalMili =0
2     timeSplitet=collectTime['listOfDifferenceTimeStamp'][i]# et række af gang
3     timeSplitetRow= timeSplitet.split("#Mellemrum#") #split de url som vi havde lagt sammen med #Mellemrum#
4     nloop=len(timeSplitetRow)
5     for j in np.arange(nloop):
6         difrenceTime= float(timeSplitetRow[j])# kaster den tal som er tidforskelt på SEC
7         totalMili += difrenceTime # lægger dem sammen
8         listofSecoun.append(difrenceTime) # så adder vi den forskel tid fra forskellige brugeren som er på sekunder
        listofTotalSec.append("{0:.3f}".format((totalMili)))# bruger kun 3 tal efter decimal
        listoflistSecoun.append(listofSecoun) # adder de forskel tider fra forskellige brugere til en liste
        listofSecoun=[] # tømmes i hver loop fordi vi vil gerne tilføje tid under en url
listOfUrlIdAllTimeStamp= pd.DataFrame(data ={'listOfUrlid':list(collectTime['listOfUrlid']),
        'listOfTimeStamp': list(listOfTotalSec)})
```

Billede 40: her beregnes total af tider som er brugt under hver URL

1	For-Loop	Størrelse af Loopen er lig med størrelse af listen af unikke url-id!
2	timeSplitet	Enhholdsvis hentes de tider som er brugt på hver url, samlet med en streng #mellemrum#!
3	timeSplitrow	Så trækkes alle tider ud fra den lange streng som har #mellemrum# i mellem!
4	Nr. 2 For-Loop	Størrelse af Loopen er antallet af tider som er mellem strengen #mellemrum#
5	totalMili	Lægger de tider som er på sekunder sammen!
6	listoftotalSec	Her bruges kun tre decimal nr. for de samlede tider og gemmes i en liste
7	listOfListSecoun	Kun de tider som er i den lange streng med en #mellemrum# vil gemmes i en liste.
8	listOfUrlIdAllTimeStamp	En Dataframe med to kolonner; en med liste af unikke URL nr. og de samlede total tid for hver URL.

Man kan se alle tidsforbrug for URL [Se Billede 41]

Her kan også ses hvilken af undersiderne brugeren har brugt meget tid

på. [Se Billede 42]

	listOfUrlid	listOfTimeStamp
0	0	1167.805
1	1	3176.229
2	2	1448.991

Billede 42: de samlet tid som er brugt under hver URL

listoflistSecoun

```
[[0.0,
  0.0,
  0.0,
  4.929,
  828.516,
  0.0,
  19.365,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
```

Billede 41: list af tider for hver url samlet på en list.

8.2.2 Tidsberegning for felter

Nedenstående billede er koden for at finde tidsforbrug for felter. [Se Billede 43]

```
1      handlinDataWithoutNone=Data[Data['handling'].notnull()]
2      handlinAndClearFieldDataWithoutNone=handlinDataWithoutNone[handlinDataWithoutNone['clearField']!='NONE']
      listOfUserId=[]
      listOfFieldName=[]
      listOfTime=[]
      listOfUrlId=[]
      listOfFeltNr=[]
      collectUrlAndFelt=[]
      #først finder vi data hvor deres handling er ikke none og bage efter grupper vi data til hver bruger id . så
      # en liste af user ider og en med felt navn, Liste af tider, Liste af url, List af felt nr, en blanding af url og felt

      lenOfListofUniqIds=len(ListOfUniqUserIds)
3      for k in np.arange(lenOfListofUniqIds):

      userId=ListOfUniqUserIds['id'][k]
      DataForHverUser= handlinAndClearFieldDataWithoutNone[handlinAndClearFieldDataWithoutNone['id']==userId]
      if(len(DataForHverUser)>1):

4          datetimeFormat = '%Y-%m-%dT%H:%M:%S.%f' # date format
5          #DataForHverUser er de data for hver bruger som vi finder de unike urler og unike felter ud af de data
6          listUniqClearUrl= pd.DataFrame(DataForHverUser['clearURL'].unique()).rename(columns={0:"urlName"})
7          listUniqClearFeldt= pd.DataFrame(DataForHverUser['clearField'].unique()).rename(columns={0:"FeildName"})

      #Med 2 for Loop så vil vi tjekke hvis En bruger har vært det samme url og samme felt
8      for x in np.arange(len(listUniqClearUrl)):

          clearUrl=listUniqClearUrl['urlName'][x] # url navn henholdsvis

          for y in np.arange(len(listUniqClearFeldt)):

              ClearFeldt=listUniqClearFeldt['FeildName'][y] #felt navn henholdsvis

9          userData= DataForHverUser[(DataForHverUser['clearURL']==clearUrl)
              & (DataForHverUser['clearField']==ClearFeldt)]#hvor data ud hvor
              #bruger har vært i en url og en felt

          lenDataUserOneLess=len(userData)-1 #LenOfdata hvor data er en liste af en bruger og en url og en felt
          if((len(userData)>1):#Hvis vi har minimum to række af data og
              #feltet er Ikke "LikeUSERID"

10             for j in np.arange(lenDataUserOneLess):

                  row1=userData.iloc[j]
                  if (row1.handling=="focus"):# tjekker hvis første rækkes handling er focus

11                     for u in (np.arange(lenDataUserOneLess-j)+j+1): # vil springe 2 række fram af
12                         # så tjekke nr 1 og 2 næste bliver 3 og 4
                        row2=userData.iloc[u]

                        if(row2.handling=="blur"):#hvis næste data rækkes handlingen er "blur"

13                         if (row1.virkStartForloebId == row2.virkStartForloebId):# tjekkes hvis både række af
                            #data har ens forløbsid
                            date1=datetime.datetime.strptime(row1.serverTimeStamp, datetimeFormat)#første rowstid
                            date2=datetime.datetime.strptime(row2.serverTimeStamp, datetimeFormat)#næste rows tid

                            diff = abs(date2-date1)# beregnes tiden når handling starter med focus og næste er blur
                            timeForHandlig = "{0:.3f}".format(diff.total_seconds())
                            numberUniqUrl= findeUrlNum(listOfuniquUrl,row1,HashlistOfUrl)# beregner tal for en url
                            numberOffeldtt=findeFeldNum(listUniqClearfield,row1,HashListFeldt,"numberOffield")
                            # finder nummer til felt

                            listOfUserId.append(row1['id']) # addes user id
                            listOfFieldName.append(row1.clearField)# addes feltnavn
                            listOfTime.append(timeForHandlig)# tiden som er blevet beregnet
                            listIOFurlId.append(numberUniqUrl) # addes url id
                            listOfFeltNr.append(numberOffeldtt)# addes felt id
                            collectUrlAndFelt.append(str(numberUniqUrl)+"-"+str(numberOffeldtt))
                            # Legges sammen url id med felt id med et strek i mellem
                        break
```

Billede 43:her findes tidsforbrug på hver felt

1	Dataframe handlinData WithoutNone	Under kolonne 'handling' står 'fokus' eller 'blur' eller 'None', og her har vi ikke brug for data som har 'None' under 'handling', for det betyder at brugeren ikke har trykket på et felt.
---	---	---

2	handlinAndClearFieldDataWithoutNone	Vi har heller ikke brug for de data som under dens 'clearField' som står 'None'. For 'None' er ikke et rigtig felt, så derfor skal vi ikke finde tidsforbrug for den, derfor sletter vi alle data men 'None' stående under 'clearField'
3	For-Loop	Længde af loop er længde af unikke bruger-id liste, for vi vil have de data der tilhører hver bruger ad gangen i Loop!
4	If-Statement	Hvis længde af data tilhørende for hver bruger er større end et, så vil programmet gå videre med analysen! Større end et, fordi i det analyse kaldes to rækker data og derfor for at undgå fejl
5	datetimeFormat	Det er tidsformat for tidsberegninger!
6	listUniqClearUrl	Her finder vi alle unikke URL'er som brugeren har besøgt
7	listUniqClearFeldt	Her finder vi alle unikke felter som brugeren har klikket på!
8&9	For-Loop i en for loop	Loop i alle de URL'er som er blevet besøgt af brugeren i rækkefølge og derefter en anden loopes i hver felt som brugeren har klikket på. Så finder programmet alle data for en bruger ad gangen som har været under en url og et felt.
10	For-Loop	Længde af loop er længde af data for hver bruger som har været under en bestemt url og på et bestemt felt
11	If-Statement	data for hver bruger som har været under en bestemt url og på et bestemt felt tjekkes hvis den første række under 'handling' er lig med 'fokus', så vil det betyde at brugeren har klikket på feltet.
12	En For-Loop i anden For-Loop	Her hvis handlingen er 'fokus' så vil programmet kigge den næste række af data, og samtidig laver ændring på den øverste for-loop, derfor næst gang den oversete loop går to rækker fremad. Det vil sige at først kigges efter række 1 og 2 data og næste gang række 3 og 4.
13	If-Statement in if-Statement	At have ens forløbs-id betyder at en bruger har været logget ind på et sammenhængende forløb. Hvis den første række data er 'fokus' og næste række data er 'blur' og samtidig har de to rækker data ens forløbs-id, så vil programmet ved at bruge tidsforskellen på to rækkerdata finde tiden som er brugt på feltet. Her kunne vi også finde unikke liste af forløbs-id først, og derefter med en loop henter vi data baseret på ens forløbs-id af gang! Herfra og frem af beregning af tid er ligesom vi har beregnet tid for hver URL!

De liste vi har oprettet for undersøgelse af tidsforbrug på hver eneste felt lægger vi sammen i en Dataframe og derefter laves en Counter af liste vi har, liste som vi har gemt feltId og samtidig gemmes url-id i! [Se Billede 44]

```

1      # en dataframe med de liste af urler, feltnavn og felttider samt de tider som er brugt til hver felt
      listOfurlAndfeildHandling = pd.DataFrame(data = {'listOfUserId': list(listOfUserId),
                                                         'listOfFeldName': list(listOfFeldName),
                                                         'listOfTime': list(listOfTime),
                                                         'listOfurlId': list(listOfurlId),
                                                         'listOfFeldNr': list(listOfFeldNr),
                                                         'collectUrlAndFeld': list(collectUrlAndFeld)})

2      #listOfurlAndfeildHandling=listOfurlAndfeildHandling.sort_values(by='listOfurlId', ascending=False)
      CounterUrl_Feld = Counter([(i) for i in listOfurlAndfeildHandling["collectUrlAndFeld"]])
      Url_Feld_Count = pd.DataFrame(data = {'url-feld': list(CounterUrl_Feld.keys()),
                                             'count': list(CounterUrl_Feld.values())})

```

Billede44:

oprettelse af en Dataframe med forskellige lister og laver Counter af en liste med felt-url id som kan ses i billede 45 i "collectUrlAndFeld"

	listOfUserId	listOfFieldName	listOfTime	listOfurlId	listOfFeltNr	collectUrlAndFelt
53	b9e50560-f9f4-11e7-8dc1-0050569e4d82	files[]	0.318	4	6	4-6
56	b9e50560-f9f4-11e7-8dc1-0050569e4d82	files[]	42.845	4	6	4-6
63	b9e50560-f9f4-11e7-8dc1-0050569e4d82	Dato for godkendelse af dokument *	1.715	4	5	4-5
62	b9e50560-f9f4-11e7-8dc1-0050569e4d82	Dato for godkendelse af dokument *	9.171	4	5	4-5
61	b9e50560-f9f4-11e7-8dc1-0050569e4d82	Dato for godkendelse af dokument *	1.960	4	5	4-5
60	b9e50560-f9f4-11e7-8dc1-0050569e4d82	Dato for godkendelse af dokument *	3.847	4	5	4-5

Billede 45: Dataframe vil indeholde brugers-id, feltnavn, tiden er brugt på feltet, UrlNr, feltNr og sidste kolonne, feltNr sammen med UrlNr med en bindestreg i mellem

Her vil finde total tid på hver felt og finde antal at gang som feltet er besøgt! [Se Billede 46]

```

listOfClearUrlNum=[]
listOfClearFeltNum=[]
listOfTotalTimeForEveryClearUrl=[]
listOfAmountOfUseEveryFeldt=[]
listOfurlFeldt=[]

1  for i in np.arange(len(uniquecollectUrlAndFelt)):# med længde af unik list af urlid-feltid
2      totalTimeOfeveryFeild=0
3      uniqUrlFelt=uniquecollectUrlAndFelt[i]
4          # så hives data ud af gang til hver unik urlid-feltid
5      DataEvryFeild= (listOfurlAndfeildHandling[listOfurlAndfeildHandling['collectUrlAndFelt']==uniqUrlFelt])
6      amount=0

7      for j in np.arange(len(DataEvryFeild)):# det er dataframe på den unik url og felt som indeholder de tider

          tt= float(DataEvryFeild['listOfTime'].iloc[j])# de tider som er blevet beregnet for hver unik felt under et bestemt url
          totalTimeOfeveryFeild+=tt # lægger de tider sammen når de er under en urlid-feltid
          amount+=1 # vil finde antal af brugte felt

          listOfAmountOfUseEveryFeldt.append(amount)
          listOfClearUrlNum.append(uniqUrlFelt.split('-')[0])
          listOfClearFeltNum.append(uniqUrlFelt.split('-')[1])
          listOfTotalTimeForEveryClearUrl.append(totalTimeOfeveryFeild)
          listOfurlFeldt.append(uniqUrlFelt)

listOfFeldtAndTotalTime = pd.DataFrame(data ={'Url-Felt': list(listOfurlFeldt),
                                             'Url number': list(listOfClearUrlNum),
                                             'Felt number': list(listOfClearFeltNum),
                                             'Total time': list(listOfTotalTimeForEveryClearUrl),
                                             'Amount of uses of each field':list(listOfAmountOfUseEveryFeldt), })

```

Billede 46: finde total tidsforbrug på hver unikke url-felt

1	For-loop	Størrelse af for-loop er størrelse af unikke list på felt-url list som indeholder felt-nummer sammen med Url-nummer
2	uniqUrlField	Henholdsvis hives ud de unikke felt-url
3	DataEvryField	Her samles alle de data som har ens felt-url
4	For-loop in For-loop	Så henholdsvis hives data ud fra listen. Listen har de ens felt-url
5	totalTimeofeveryField	Vil tiderne lægges sammen for at finde total tid for hver unikke felt-url
6	amount	Her vil lægges et tallet på hver gang en felt er blevet besøgt, dette gøres for at tælle.
7	listOfFeldtAndTotalTime	Her alle de lister vi har oprettet slås sammen på en Dataframe. I kan se Dataframe på billede 44.

Her kan i se total tidsforbrug på hver unikke Url-Felt og antal gang den unikke Url-Felt er besøgt!

[Se Billede 47]

	Url-Felt	Url number	Felt number	Total time	Amount of uses of each field
0	4-10	4	10	5.986	1
1	4-14	4	14	3.173	1
2	4-16	4	16	10.444	1
3	3-13	3	13	4.169	1
4	4-7	4	7	20.482	6

Billede 47: her kan i se det total tidsforbrug for hver unikke Url-Felt og antal gang som brugeren har nyttet Feltet

8.2.3 Gruppering af bruger

Her vil fortælles hvordan vi har oprettet forskellige gruppebruger som vi skal lave alle ovenstående analyse på RID som er resursers-id og PID som er persons-id. De blev deles op 14 forskellig grupper som starter med gruppe nul.

```
def makeDataInCsv(Data,i):
    data_sluts[]
    csvnames=""
    if is0:
        csvnames='gennem_RID_enscvr.csv'
        completedDurations Data[Data['forloebVarighed'].notnull()]
        ridCompletedDuration=completedDuration[\"RID\"].astype(str).str.isdigit()#hvor i rid column er tal
        try:
            data_sluts ridCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is1:
        csvnames='gennem_RID_ikke_enscvr.csv'
        completedDurations Data[Data['forloebVarighed'].notnull()]
        ridCompletedDuration=completedDuration[\"RID\"].astype(str).str.isdigit()
        try:
            data_sluts ridCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is2:
        csvnames='gennem_RID_PID_enscvr.csv'
        try:
            completedDurations Data[Data['forloebVarighed'].notnull()]
            pidCompletedDuration=completedDuration[\"PID\"].astype(str).str.contains(\"-\")#hvor vi har personid med(\"-\") i
            data_sluts pidCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is3:
        csvnames='gennem_RID_PID_ikke_enscvr.csv'
        completedDurations Data[Data['forloebVarighed'].notnull()]
        pidCompletedDuration=completedDuration[\"PID\"].astype(str).str.contains(\"-\")
        try:
            data_sluts pidCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is4:
        csvnames='gennem_PID_enscvr.csv'
        completedDurations Data[Data['forloebVarighed'].notnull()]
        pidCompletedDuration=completedDuration[\"PID\"].notnull()
        try:
            data_sluts pidCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is5:
        csvnames='gennem_PID_ikke_enscvr.csv'
        completedDurations Data[Data['forloebVarighed'].notnull()]
        pidCompletedDuration=completedDuration[\"PID\"].notnull()
        try:
            data_sluts pidCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is6:
        csvnames='ikke_gennem_RID_enscvr.csv'
        notCompletedDurations Data[Data['forloebVarighed'].isnull()]
        ridCompletedDuration=notCompletedDuration[\"RID\"].astype(str).str.isdigit()#hvor i rid c
        try:
            data_sluts ridCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is7:
        csvnames='ikke_gennem_RID_ikke_enscvr.csv'
        notCompletedDurations Data[Data['forloebVarighed'].isnull()]
        ridCompletedDuration=notCompletedDuration[\"RID\"].astype(str).str.isdigit()#hvor i rid column er tal
        try:
            data_sluts ridCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is8:
        csvnames='ikke_gennem_RID_PID_enscvr.csv'
        notCompletedDurations Data[Data['forloebVarighed'].isnull()]
        ridCompletedDuration=notCompletedDuration[\"RID\"].astype(str).str.contains(\"-\")#hvor i rid column er tal
        try:
            data_sluts ridCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is9:
        csvnames='ikke_gennem_RID_PID_ikke_enscvr.csv'
        notCompletedDurations Data[Data['forloebVarighed'].isnull()]
        ridCompletedDuration=notCompletedDuration[\"RID\"].astype(str).str.contains(\"-\")#hvor i rid column er tal
        try:
            data_sluts ridCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is10:
        csvnames='ikke_gennem_PID_enscvr.csv'
        notCompletedDurations Data[Data['forloebVarighed'].isnull()]
        pidCompletedDuration=notCompletedDuration[\"PID\"].notnull()#hvor i rid column er tal
        try:
            data_sluts pidCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is11:
        csvnames='ikke_gennem_PID_ikke_enscvr.csv'
        notCompletedDurations Data[Data['forloebVarighed'].isnull()]
        pidCompletedDuration=notCompletedDuration[\"PID\"].notnull()#hvor i rid column er tal
        try:
            data_sluts pidCompletedDuration.query('cvr=cvrIndberet')
        except:
            print(i)
        return data_slut, csvname
    elif is12:
        csvnames='hele_Data_gennem.csv'
        data_sluts Data[Data['forloebVarighed'].notnull()]
        return data_slut, csvname
    elif is13:
        csvnames='hele_Data_ikke_gennem.csv'
        data_sluts Data[Data['forloebVarighed'].isnull()]
        return data_slut, csvname
#----- finde urlrejsen hver unik id i data_slut, den del er kopi fra AnalyseData
```

Gruppe 0	'gennem_RID_enscvr.csv'	Samler de data som forloebVarighed er ikke null og de brugere gennemgået forløbet, og derefter ud af de data samles de data som har et tal stående under RID og til sidst samles de data som deres cvr er lige med cvrIndberettet!
Gruppe 1	'gennem_RID_ikke_enscvr.csv'	Det ligner første gruppe, bortset fra her de data som brugerens cvr er ikke lige med cvrIndberettet!
Gruppe 2	'gennem_RID_PID_enscvr.csv'	De bruger er gennemgået forløbet, tjekker hvis RID er ikke et tal og har en bindestreg på sig, som det gøre at RID ligner PID og samler de data som cvr er lige med cvrIndberettet!
Gruppe 3	'gennem_RID_PID_ikke_enscvr.csv'	De bruger er gennemgået forløbet, tjekker hvis RID har en bindestreg på sig og cvr er ikke lige med cvrIndberettet!
Gruppe 4	'gennem_PID_enscvr.csv'	De bruger er gennemgået forløbet, samler de data som under deres PID ikke er null og cvr er lige med cvrIndberettet!
Gruppe 5	'gennem_PID_ikke_enscvr.csv'	De bruger er gennemgået forløbet, og samler de data som under PID er ikke tom og cvr er ikke lige med cvrIndberettet!
Gruppe 6	'ikke_gennem_RID_enscvr.csv'	De bruger er nul i forløbVarighed, dvs. dem der ikke er gennemgået forløbet, samler de data som under deres RID står et tal og cvr er lige med cvrIndberettet!
Gruppe 7	'ikke_gennem_RID_ikke_enscvr.csv'	Samles dem der ikke er gennemgået forløbet, samler de data som under RID står et tal og cvr er ikke lige med cvrIndberettet!
Gruppe 8	'ikke_gennem_RID_PID_enscvr.csv'	Samler de data der ikke er gennemgået forløbet, og de data deres RID har en bindestreg på sig og cvr er lige med cvrIndberettet!
Gruppe 9	'ikke_gennem_RID_PID_ikke_enscvr.csv'	Samler de data der ikke er gennemgået forløbet, de data som deres RID indeholder en bindestreg og cvr er ikke lige med cvrIndberettet!
Gruppe 10	'ikke_gennem_PID_enscvr.csv'	Dem der ikke er gennemgået forløbet, samler de data som under PID er ikke null og cvr er lige med cvrIndberettet!
Gruppe 11	'ikke_gennem_PID_ikke_enscvr.csv'	Samler de data der ikke er gennemgået forløbet, og under PID er ikke tom og cvr er ikke lige med cvrIndberettet!
Gruppe 12	'hele_Data_gennem.csv'	Alle de bruger er gennemgået forløbet.
Gruppe 13	'hele_Data_ikke_gennem.csv'	Alle de bruger er ikke gennemgået forløbet.

Kombinationen af de to grupper, 12 og 13 har alle bruger på sig.

8.2.4 Oprettelse af histogram

```
def histogram_For_EveryFelt_in_EveryURL_Save_Quatiler(listOfurlAndfeildHandling,
collectUrlAndFelt,listUniqueClearfeild,HashListfeild,path,quatiler,csvName):
    #Lave histogram for hver de unikke urler and felt og tider som brugt per url og felt
    t50_arr=[]
    t90_arr=[]
    Count=[]
    #ListOfurlAndfeildHandling=ListOfurlAndfeildHandling.sort_values(by='collectUrlAndFelt', ascending=False)
    uniqcollectUrlAndFelt= listOfurlAndfeildHandling['{}'].format(collectUrlAndFelt)
    #uniqcollectUrlAndFelt=uniqcollectUrlAndFelt.sort_index()
    uniqUrlFElt=""
    lessOf5Url_felt_index=[]
    if uniqcollectUrlAndFelt is not None:
        for d in np.arange(len(uniqcollectUrlAndFelt)):# List of unikke url o felt
            uniqUrlFElt=uniqcollectUrlAndFelt[d]

            DataEvryFeild= (listOfurlAndfeildHandling[listOfurlAndfeildHandling['collectUrlAndFelt']==uniqUrlFElt])
            urlNumber=uniqUrlFElt.split('-')[0]
            feildtName = format_filename(str(findeHASHFeildName(int(uniqUrlFElt.split('-')[1]),listUniqueClearfeild,HashListfeild)))
            t50=-1
            t90=-1
            if (len(DataEvryFeild.listOfTime) > 5): # hvis der er mere end 5 hits så bliver der oprettes histogram af den
                t50=pd.to_numeric(DataEvryFeild.listOfTime).quantile(q=.5)
                t90=pd.to_numeric(DataEvryFeild.listOfTime).quantile(q=.9)
                plt.figure(figsize=(10,5))
                plt.hist(pd.to_numeric(DataEvryFeild['listOfTime']),range=[0,60],bins=31) # List af tider i hver uni
            try :
                plt.title( feildtName+"\n"+ url - felt: "+ uniqUrlFElt +"\n" +
                    ". fraktil 50-90 : "+ str("{0:.3f}".format(t50))+ " "+str("{0:.3f}".format(t90)))
            except:
                plt.title( feildtName+"\n"+ url - felt: "+ uniqUrlFElt +"\n" +
                    ". fraktil 50-90 : "+ str("{0:.3f}".format(t50))+ " "+str("{0:.3f}".format(t90)))
            plt.axvline(t50, color='red')
            plt.axvline(t90, color='yellow')
            plt.xlabel('Time (Min)')
            plt.ylabel('Amount ')
            plt.xticks(rotation=45)
            #oprattes hver histogram png med felt navn og tider og gemmes i nedstående sti
            plt.savefig(path+csvName+'_'+feildtName+'_'+FeltNumber_+str(uniqUrlFElt.split('-')[1])+ '_UrlNumber_'+ str(urlNumber)+'.png')
            t50_arr.append(t50)
            t90_arr.append(t90)
            plt.clf()
            plt.close()# Close a figure window
            Count.append(len(DataEvryFeild.listOfTime))
        else:
            lessOf5Url_felt_index.append(d)
            uniqcollectUrlAndFelt=uniqcollectUrlAndFelt.drop(lessOf5Url_felt_index)
    else:
        print('uniqcollectUrlAndFelt is empty')
    if uniqcollectUrlAndFelt is not None:
        t_quatiler=pd.DataFrame({"URL-Felt": uniqcollectUrlAndFelt, "t50":t50_arr, "t90":t90_arr, "Count":Count})
        t_quatiler.to_csv("Luk_Virkksomhed/Quatiler/"+str('{}').format(quatiler))+'.csv',sep=";", index=False)
    else:
        print('uniqcollectUrlAndFelt after delete som of index is empty')
    return uniqcollectUrlAndFelt,t_quatiler
```

1	If-statement 1	Hvis liste felt-url ikke er tom så vil programmet gå videre ellers i Else-statement vil printes at listen er tom. Med den if-statement forhindre fejl når listen er tom!
2	For-Loop	Loop størrelse er længde af felt-url liste. Felt-url liste har felter med en bindestreg sammen med dens tilhørende url tallet.
3	DataEvryFeild	Finder alle data som har besøgt en bestemt url-felt
4	urlNumber og feildtName	Ud fra liste felt-url id, finder programmet navnet for hver feltet og standardisere navnet!
5	T50 og t90	Initialisere to variabler for at beregne fraktil. Fraktil er en vis andel af et observationssæt. Her vil vi beregne fraktil for 50% og 90%.
6	If-statement	Hvis der er mere end fem personer har besøgt feltet så vil programmet oprette en histogram for feltet.

7	Plt.hist	Converteres en list af tidsforbruget til tal og lave histogram.
8	Plt.title	Laves en titlen På hver histogram som vil informere, hvilken felt den histogram for, hvilken url tilhøres histogram for og dens fraktil undersøgelses.
9	Plt.savefig	Her vil histogrammet gemmes som en png-fil med feltnavet, og derefter lægges ind stien som vi oplyser!
10	Plt.clf Plt.close	Vil lukkes histogrammet efter at histogram bliver gemt.
11	Else-statement	Hvis data er mindre end fem, dvs. en felt som har mindre af fem besøg, vil programmet gemme indeks. Til sidst slettes alle de data fra som har de indeks.
12	If-statement	Hvis efter at slette de indeks, er listen ikke er tom, så vil oprettes en Dataframe med de data, og gemmes som en Csv i stien.

9) Django

For at lave en selvstændig applikation har jeg valgt at bruge Django. Django er en Python-webramme, der bruges til at udvikle webapplikationer. Django er en Python-baseret gratis og open source webramme, der følger model-view-controller arkitektoniske mønster.

For at oprette Django skal vi først installere en virtualenv. En virtualenv adskiller Python-kode fra Windows-operativsystemet og oprette en isoleret miljø som kørsel af Python kode er det anderledes af kørsel af Python kode på Windows.

Django adresse er: <http://127.0.0.1:8000>

9.1. Installation af Virtualenv og Django

For at installere virtualenv skal vi have installeret Python 3 version først og derefter ind i Gitbash skrives disse script. [13]

- Cd myproject
- pip3 install virtualenv
- Cd .venv
- In Linux -> source ./Scripts/activate (ind i terminal cd Scripts activate.bat)
- pip3 install Django! I det projekt er jeg brugt pip3 install Django==2.2.5
- Python manage.py startapp <navn for projektet> så vil et projekt blive oprettet
- Python manage.py makemigrations
- Python manage.py migrate (Ændringer på appen vil opdateres efter ændring i 'models.py')
- Python manage.py createsuperuser (har givet username: admin password: mah123456)

- Python manage.py runserver (Django vil køre på: <http://127.0.0.1:8080/>)

De overstående installerede programmer vil så blive lagt under venv-fil. [14]

For at en anden server kan køre programmet, man kan installere de moduler som en plugin ind i en txt fil med at indtaste script i root af projektet:

```
pip freeze > requirements.txt
```

For at installere alle de scripte, kan det skrives:

```
pip install -r requirements.txt
```

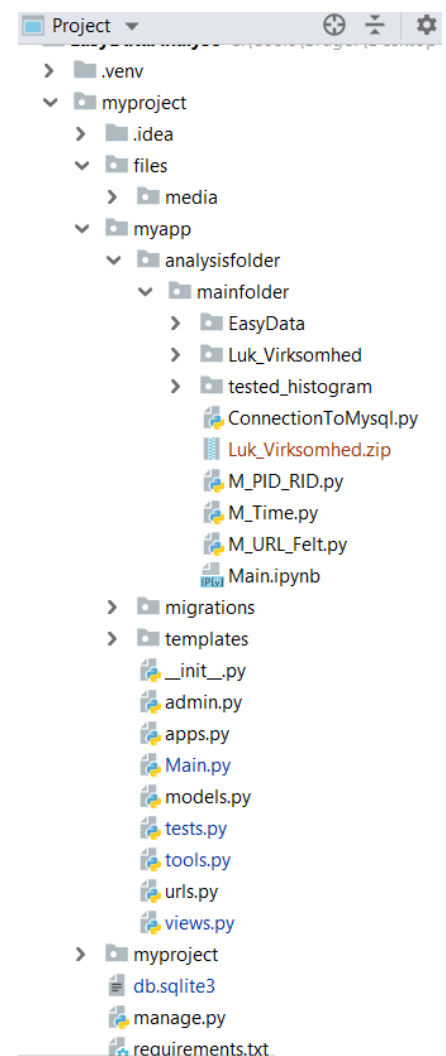
9.2 Django design

Mappen for min Django indeholder myproject

9.2.1 Controller

- **Urls.py:** For oprette views på Django er vi først nødt til i urls.py under 'myapp' og 'myproject' filer at informere om hvordan siderne på views navigeres.
- **'myapp':** Her har jeg lagt alle moduler jeg har oprettet for analysen under filen mainfolder 'myapp' i Django projektet. Efter at oprette myapp skal jeg informere Django om at denne applikation eksisterer. For at oplyse Django skal vi i setting under INSTALLED_APPS tilføjes 'myapp', så vil Django også kende 'myapp'. [Se billede 'myapp']

Under 'Main.py' ligger alle kommandoer der vil kalde alle metoder for analysen. Jeg har lagt alle de kommandoer ind i en metode med en variabel. Variabel er en Csv fil som brugeren selv kan give som en input og Csv sendes til 'myapp' gennem Django applikationen.



Man kan i Main.py se alle kommandoer som der køres analyse på i den nedenstående billede.

```
def do_algorithm(myinput, mode):
    # rootconn = connection.dockerConnection()
    # Data= pd.DataFrame(connection.sqlQuery("SELECT * FROM Luk_Virksomhed",rootconn)) #TODO

    main_Data= get_data(myinput, mode)
    Data = main_Data.copy()
    Data = Data.rename(
        columns={0: 'id', 1: 'forloebVarighed', 2: 'virkStartForloebsId', 3: 'cvr', 4: 'cvrIndberet', 5: 'RID',
                  6: 'PID', 7: 'serverTimeStamp', 8: 'formURL', 9: 'feltType', 10: 'feltId', 11: 'feltNavn',
                  12: 'handling'})

    print('-----\\|||||n\n\n\n\n--\n\n\n', type(Data), Data.keys())
    print('+++\\|||||n\n\n\n\n--\n\n\n', type(main_Data), main_Data.keys())

    Data,listOfunigUrl,HashlistOfUrl,listUnigeClearfield,HashListfeild,ListOfUniqUserIds = time.make_ready_new_columns(Data)

    unigIdGruppeClearUrl = felt.findURLRejser(Data,"id","clearURL","newClearURL")

    unigIdGruppeClearUrl = felt.enterNumbersInsteadOfName(unigIdGruppeClearUrl,HashlistOfUrl,listOfunigUrl)

    unigIdGruppeClearUrl = felt.findURLRejser(Data,"id","clearURL","newClearURL")

    unigIdGruppeClearUrl = felt.enterNumbersInsteadOfName(unigIdGruppeClearUrl,HashlistOfUrl,listOfunigUrl)

    UrlRejserAndCount = felt.makecounter(unigIdGruppeClearUrl,"newClearURL")

    felt.saveToCsv_2(UrlRejserAndCount,'ListOfUrlRejser','count',"myapp/analysisfolder/mainfolder/Luk_Virksomhed/Sunburst/Alle_Bruger/Unique_UrlRejser_&_Count.csv")

    felt.saveToCsv_2(listOfunigUrl,'urlName','urlNumber',"myapp/analysisfolder/mainfolder/Luk_Virksomhed/UniqueList/UniqueUrl.csv")

    felt.saveToCsv_2(listUnigeClearfield,'FeildName','numberOfField',"myapp/analysisfolder/mainfolder/Luk_Virksomhed/UniqueList/UniqueFeildList.csv")

    # rootconn = connection.dockerConnection()
    # Data= pd.DataFrame(connection.sqlQuery("SELECT * FROM Luk_Virksomhed",rootconn))

    Data = main_Data.copy()
    Data = Data.rename(
        columns={0: 'id', 1: 'forloebVarighed', 2: 'virkStartForloebsId', 3: 'cvr', 4: 'cvrIndberet', 5: 'RID',
                  6: 'PID', 7: 'serverTimeStamp', 8: 'formURL', 9: 'feltType', 10: 'feltId', 11: 'feltNavn',
                  12: 'handling'})
    Data= felt.delete_Handlig_Bluer_And_Reset_Index(Data)

    Data, listOfunigUrl ,HashlistOfUrl ,listUnigeClearfield, HashListfeild, ListOfUniqUserIds = time.make_ready_new_columns(Data)

    listOfUrIdAndFeltRejse = felt.findeFeildOrServerTime(Data,listOfunigUrl,HashlistOfUrl,'clearField')

    listOfUrIdAndFeltRejse = felt.enters_Numbers_Instead_Of_Felt_Name(listOfUrIdAndFeltRejse,HashListfeild)

    felt.make_Csv_For_FeldList_for_every_URL(listOfunigUrl,listOfUrIdAndFeltRejse,
                                              "myapp/analysisfolder/mainfolder/Luk_Virksomhed/Sunburst/Alle_Bruger/Unique_Felt_Rejser_And_Count_Every_URL/")

    Data = main_Data.copy()
    Data = Data.rename(
        columns={0: 'id', 1: 'forloebVarighed', 2: 'virkStartForloebsId', 3: 'cvr', 4: 'cvrIndberet', 5: 'RID',
                  6: 'PID', 7: 'serverTimeStamp', 8: 'formURL', 9: 'feltType', 10: 'feltId', 11: 'feltNavn',
                  12: 'handling'})

    Data,listOfunigUrl,HashlistOfUrl,listUnigeClearfield,HashListfeild,ListOfUniqUserIds = time.make_ready_new_columns(Data)

    listOfurIdAndServerTimeStamp=felt.findeFeildOrServerTime(Data, listOfunigUrl, HashlistOfUrl, 'serverTimeStamp')

    listOfurIdAndDiffrenceTimeStamp=time.calculateTime_URL(listOfurIdAndServerTimeStamp,listOfurIdAndServerTimeStamp['UrId'])

    listOfurIdAllTimeStamp,listoflistSecoun=time.calculateTime_total_URL(listOfurIdAndDiffrenceTimeStamp)

    time.make_Histogram_For_EveryURL(listoflistSecoun,'Alle_Bruger',"")

    listOfurIdAndfeildHandling,Url_Felt_Count=time.felddtTimeCalculate( Data,ListOfUniqUserIds,listOfunigUrl,HashlistOfUrl,listUnigeClearfield,HashListfeild)

    unigcollectUrlAndFelt,t_quatil=time.histogram_For_EveryFelt_in_EveryURL_Save_Quatiler(listOfurIdAndfeildHandling,
                                              'collectUrlAndFelt', listUnigeClearfield, HashListfeild,"myapp/analysisfolder/mainfolder/Luk_Virksomhed/Histogram/Alle_Bruger/Felter_Tidsforbrug/",'quatiler',"")

    Data = main_Data.copy()
    Data = Data.rename(
        columns={0: 'id', 1: 'forloebVarighed', 2: 'virkStartForloebsId', 3: 'cvr', 4: 'cvrIndberet', 5: 'RID',
                  6: 'PID', 7: 'serverTimeStamp', 8: 'formURL', 9: 'feltType', 10: 'feltId', 11: 'feltNavn',
                  12: 'handling'})

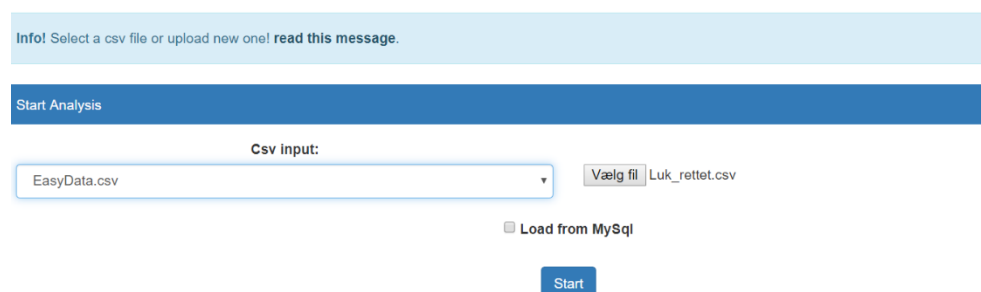
    PID RID.make Analyse For PID RID(Data, 14, 'quatiler PID RID', 'Gruppet Bruget PID RID')
```

9.2.2 Frontend (Views)

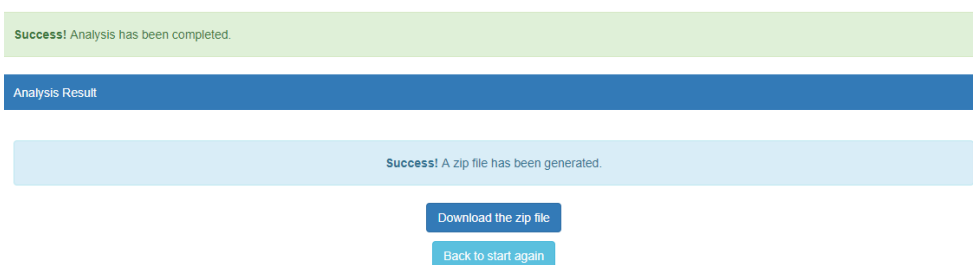
Ind i Views har vi to metoder der gør at brugeren kan se de nedenstående views!

- Når Django køre den første metode sender en list af Csv til "index.html" under en drop-down menu, og derefter med post metode sendes navnet til analysen. Der er også mulighed at brugeren vælger en Csv og giver denne som en input og de importerede data gemmes i Django database gennem "models". Der også mulighed at vælge data fra MySQL in Docker. Man skal først med kommandoer gemme Csv fil in MySQL in Docker.
Csv-filer for drop-down-menu under denne sti: myproject\myapp\analysisfolder\mainfolder\EasyData".
- Anden metode vil programmet finde Csv som brugeren har sendt og derefter lave analyse på den og gemme analysens svar som en zip-fil (som består af forskellige Csv-filer og histogram png-filer). Efter analysen er færdig bliver zip-filen gemt ind i sti under 'myproject/files/media' og derefter sendt på "result.html"! Der er også mulighed for at gemme Csv filen på skrivebord!

Nedenfor kan man se hvordan den Django app views, jeg har oprettet, ser ud. [Se Billede 48&49]



Billede:48 Django analyse projektet forside. Mulighed for at vælge en Csv-fil fra computeren eller fra menu eller læse data fra MySQL i Docker.



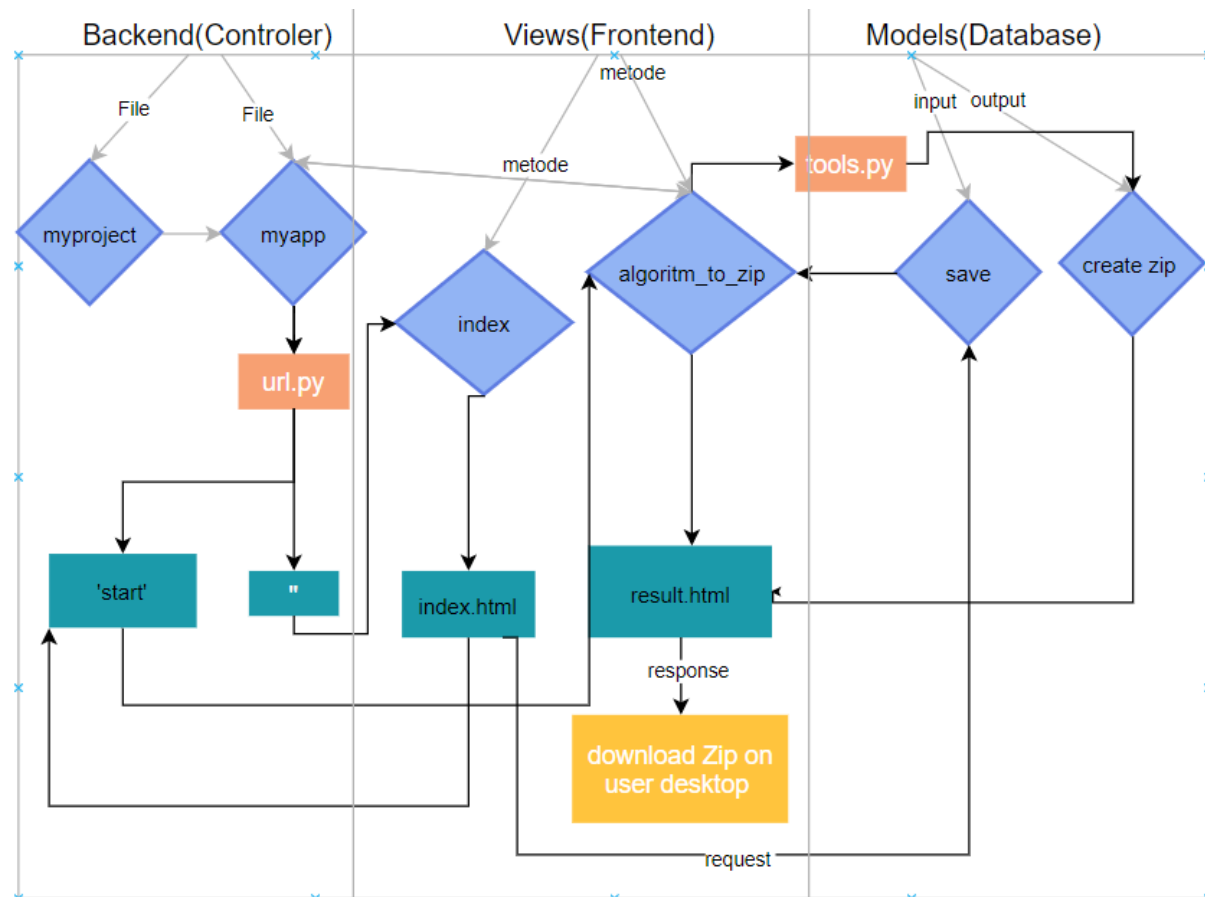
Billede 49: med et tryk på knappen "Download the zip file" kan gemme som en zip-fil på computeren

Efter at trykke på start vil analysen køres, og så efter kørselstid vil der dukke en anden side op viser en zip-fil.

Zip-filen indeholder alle Csv-filer og histograms png-filer som er blevet dannet efter analysen kørt.

9.2.3 Django Database (Models)

De data som bliver sendt fra brugeren og retures til brugeren gemmes in Django databasen! I models.py har vi oprettet to variabler "name" og en "file" som input dataene vil blive gemt under. Der er også en sti hvor filerne skal gemmes, stien kan ændres under setting.py. Man kan se hvordan mit Django projekt er bundet sammen. [Se Billede 50]



Billede 50: flowchart af Django app

Django System Architecture

10) Test

Jeg har oprettet test som vil teste png-filer som oprettes gennem Histogram. Først har jeg lagt png-filerne manuelt under stien: `myapp/analysisfolder/mainfolder/tested_histogram/`

Så vil programmet samlinger sammen med de png-filer som programmet opretter efter at testen køres. For at lave testen så har jeg oprettet to metoder først.

```
def get_histogram_tested():
    hlist = []
    hlist = []
    counter = 0
    while(True):
        try:
            im = Image.open(f"myapp/analysisfolder/mainfolder/tested_histogram/_TimeUsedforUrl_{counter}.png")
            counter += 1
            hlist += [im]
        except:
            break
    return hlist

def get_histogram_generated():
    counter = 0
    hlist = []
    while(True):
        try:
            im = Image.open(f"myapp/analysisfolder/mainfolder/Luk_Virksomhed/Histogram/Alle_Bruger/URL_Tidsforbrug/_TimeUsedforUrl_{counter}.png")
            counter += 1
            hlist += [im]
        except:
            break
    return hlist
```

Billede 51: under de adresse er png-filer som skal sammenlignes med hinanden

Derefter kalder jeg dem i den nedenstående metode ind i class "HistogramTestCase".

```
class HistogramTestCase(TestCase):
    def test_analysis_histogram_count(self): # it must start with test
        """test that two nums add together"""
        myinput = {}
        myinput["csvSelected"] = 'myapp/analysisfolder/mainfolder/EasyData/EasyData.csv'

        do_algorithm(myinput)
        tested_histograms = get_histogram_tested()
        generated_histograms = get_histogram_generated()
        #print("len(tested_histograms):", len(tested_histograms))
        #print("len(generated_histograms)", len(generated_histograms))
        self.assertEqual(len(tested_histograms), len(generated_histograms))

        # a = ImageChops.difference(hlist[0], image2)

        # self.assertEqual(, )
    def test_analysis_histogram(self):
        tested_histograms = get_histogram_tested()
        generated_histograms = get_histogram_generated()
        for i in range(len(tested_histograms)):
            this_tested = tested_histograms[i]
            this_generated = generated_histograms[i]
            diff1 = ImageChops.difference(this_tested, this_generated)
            #print("a", diff1.getextrema())
            expected_extrema = ((0, 0), (0, 0), (0, 0), (0, 0))
            self.assertEqual(diff1.getextrema(), expected_extrema)
```

Billede 52: kigges på forskellen af to billeder af gang og forskellet skal være lig med expected_extrema

I først testmetode tjekkes antallet af png-filer, er ens, de png-filer som ligger i de ovenstående stier.
[Se Billede 52]

Næste testmetode tjekkes, hvis hver png-fil fra en fil er ens med den anden png-fil i som ligger i en anden fil. De tjekkes i rækkefølgen som de er blevet gemt. Der er en metode som finder forskellen mellem to billeder og returnerer svart sådan format stående på variable "expected_extrema"! Hvis de er det samme så vil testen passes. [Se Billede 52]

For at testen kan køres, først burde man køre programmet med en bestemt antal af data, og derefter tilføjer de png-filer, som er blevet oprettet, under filen 'tested_histogram'. Med bestemt antal af data kører vi testen, med at indtaste det nedenstående script i terminal:

- `python manage.py test`

Så vil der bliver printet at testen er kørt!

[Se Billede 53]

```
System check identified no issues (0 silenced).
a ((0, 0), (0, 0), (0, 0), (0, 0))
a ((0, 0), (0, 0), (0, 0), (0, 0))
a ((0, 0), (0, 0), (0, 0), (0, 0))
a ((0, 0), (0, 0), (0, 0), (0, 0))

len(tested_histograms): 4
len(generated_histograms) 4
.
-----
Ran 2 tests in 2.646s
```

Billede 53: når testen kørt vil printes dette.

11) MySQL ind Docker

Jeg har også lagt data i MySQL ind i Docker container gennem phpMyAdmin. Med de nedenstående kode in Gitbash når Docker kører på maskine:

- `docker stop $(docker ps -a -q)`
- `docker rm $(docker ps -a -q)`
- `docker container ls -a`

Jeg har oprettet en docker-compose.yml fil for at hente MySQL og phpmyadmin images med de stående info: [Se

Billede 54]

Ind i Gitbash skrives

- `docker-compose up`

Man kan med en browser-baseret og en bedre brugergrænseflade phpMyAdmin nemt importere og Csv-fil ind i MySQL. Man kan se phpMyAdmin på adressen:[10]

<http://127.0.0.1:8080>

Når vil genstart docker-compose up, for at data som ligger i MySQL, ikke bliver slettet, skriver man dette:

- `docker-compose up -d --no-recreate`

Jeg har oprettet databasen som kan ses i den nedenstående billede. [Se Billede 55]

```
version: '3.1'
services:
  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: Ervervsstyrelse
    ports:
      - "3306:3306"
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    restart: always
    environment:
      PMA_HOST: db
      PMA_USER: root
      PMA_PASSWORD: root
    ports:
      - "8080:80"
```

Billede 54: docker-compose.yml fil, her hentes Docker og

id	forloebVarighed	virkStartForloebId	cvr	cvrIndberet	RID	PID	serverTid
e3ffda0-ee8b-11e7-b724-0050569e4d82	forloebVarighed	1dfa6a1-a709-4c03-acbf-e02469854658	1	cvrIndberet	RID	PID	2018-01-01T00:37
e3ffda0-ee8b-11e7-b724-0050569e4d82	forloebVarighed	1dfa6a1-a709-4c03-acbf-e02469854658	1	cvrIndberet	RID	PID	2018-01-01T00:37

Billede 55: phpMyAdmin

For at forbinde til MySQL-database gennem Docker med Python, skrev jeg de nedenstående metoder.

[Se Billede 56]

```
import sys
import mysql.connector
def dockerConnection():
    rootconn = myconnect('root','root')
    return rootconn
def myconnect(user, pw):
    conn = mysql.connector.connect(host='localhost',
                                   database='Ervervsstyrelse',
                                   user='root',
                                   password='root')
    conn.autocommit = True
    return conn
def sqlQuery(sqlString, conn):
    try:
        cursor = conn.cursor()
        cursor.execute(sqlString)
        res = cursor.fetchall()
        return res
    except Exception as ex:
        print(str(ex), file=sys.stderr)
    finally:
        cursor.close()
def sqlDo(sqlString, conn):
    try:
        cursor = conn.cursor()
        cursor.execute(sqlString)
        res = cursor.fetchallwarnings()
        return res
    except Exception as ex:
        print(str(ex), file=sys.stderr)
    finally:
        cursor.close()
"Done"
```

Billede56: oprettelse til MYSQL med at brug Docker

Derefter vil kaldes de ovenstående metode gennem den nedenstående metode og derefter vil dataene hentes og gemmes under 'Data'. [Se Billede 57]

```
def get_data(myinput, mode):
    if mode == "mysql":
        from .analysisfolder.mainfolder import ConnectionToMysql as connection
        rootconn = connection.dockerConnection()
        Data = pd.DataFrame(connection.sqlQuery("SELECT * FROM Luk_Virksomhed", rootconn))
    else:
        Data = pd.read_csv(myinput["csvSelected"], sep=';', dtype='str')
    return Data
```

Billede57a: oprettelse til MYSQL og hiver data ud

```
Data = Data.rename(
    columns={0: 'id', 1: 'forloebVarighed', 2: 'virkStartForloebId', 3: 'cvr', 4: 'cvrIndberet', 5: 'RID',
             6: 'PID', 7: 'serverTimeStamp', 8: 'formURL', 9: 'feltType', 10: 'feltid', 11: 'feltNavn',
             12: 'handling'})
```

Billede57b: navngivning af koloner

12) Sunburst kode

For at oprette Sunburst bruger vi de Csv-filer vi har oprettet før og lagt under Sunburst. De Csv-filer har to lister, en liste som indeholder list af med de URL-BrugerRejser eller felt-BrugerRejser og list af deres tæller(Counter).

For at lave Sunburst hentes Csv-fil som vi skal bruge. Hvis der er flere Csv-fil så vil kalde dem i en Loop. I den nedenstående kode kan i se hvordan de Csv-filer danner Sunburst.

```
1  sekvens, ufId = klaus211('Start_Virkksomhed/UniqueUrlRejserAndCountWithoutHttp.csv')
   #print(" helllloo", sekvens)
2  farve_css, text_css = farver(ufId)
3  sunburst(sekvens, farve_css, text_css, size=500)
```

1	Sekvens, ufld	Csv sendes til en metode som er inspireret fra Sunburst. Metoden vil returnere to liste, en liste baseret på de navnene opretter antal farver og en list baseret på antal af dem. på antallet af sager vil vise massen og en liste der vil informerer på hvilken sekvens de skal stå.
2	farve_css , text_css	Listen med navne og antal af dem vil oprette farve og masse
3	sunburst	Tre lister, en har farve for navne og antal af de farver og en list der viser i hvilken sekvens de skal stå

Koden for Sunburst originalt er inspireret fra: <https://bl.ocks.org/kerryrodden/7090426>

Som er selvfølgelig tilpasset koden til denne behov, som er visning af URL-BrugerRejser og felt-BrugerRejser.

13) Programmetts kørselstid

Analyse af data tager tid. For at vi kan estimere programmets udførelsestid har vi brugt 'time' og ved hjælp af 'print()' kan jeg se hvor lang tid programmet tager til at udføre hver stykke af kode.

For at finde ud af kørselstid for hver stykke kode kan vi bruge 'time'. Før en metode kan skrives: tstart=time.time(), og derefter bruger vi den tslut=time.time() for at printe tiden ud for programmets kørselstid.

Jeg har skrevet mange print-line for at observere hvordan kørselstiden skalerer sig især på PID og RID for at se hvor længe programmet er kørt. Man kan se i det nedenstående hvad de bliver printet ud efter at køre PID og RID analyse!

Data med 100000 række data	Data med 200000 række data
Gruppe bruger nr:0 Time used: 0.001 Min	Gruppe bruger nr:0 Time used: 0.005 Min
Gruppe bruger nr:1 Time used: 4.644 Min	Gruppe bruger nr:1 Time used : 12.924 Min
Gruppe bruger nr:2 Time used: 0.001 Min	Gruppe bruger nr:2 Time used : 0.002 Min
Gruppe bruger nr:3 Time used: 0.944 Min	Gruppe bruger nr:3 Time used : 2.511 Min

...	...
Gruppe bruger nr:11 Time used: 21.758 Min	Gruppe bruger nr:11 Time used : 101.938 Min
Gruppe bruger nr:12 Time used: 68.896 Min	Gruppe bruger nr:12 Time used : 342.918 Min

14) Konklusion

Ved at analysere de data der kommer fra en hjemmeside, kan man få et kendskab til hjemmesiden uden man overhovedet kender den i forvejen. Med analyse kan man se hvor mange undersider der er i alt, og hvor mange og hvilke felter der er på hver underside, og i hvilken rækkefølge, hvor lang tid der er blevet brugt på hver underside og på hvert felt i en periode, og hvor mange brugere der har besøgt Virk.dk. Alt hvad brugere har foretaget sig på Virk.dk kan registreres og kan bruges til analysen. Med analysen opnår man således mange ting.

Analyse af store dataset tager lang tid og mange fejl dukker først op, når man har kørt hele data. I dette tilfælde vidste vi ikke fra starten af hvilke feltnavne der er. Disse feltnavne skulle bruges til at navngive filer! Det var ikke nemt at analysere, for der var mange lange navne med forskellige tegn i og også andre sprog. Vi var nødt til at beholde de feltnavne som var på andre sprog for en bedre analyse. Til sidst kunne jeg løse problemet ved at jeg fandt en metode hvor vi nemt kunne normalisere navnene for navngivning af filer!

Når jeg kørte programmet for alle data på én gang, brugte analysen så meget af computerens kapaciteten at man ikke samtidig kunne bruge computeren til andre analyser, da jeg ellers fik fejlen 'manglede hukommelse'. Derfor lod vi programmet køre på andre servere end min computer, så jeg kunne arbejde videre på projektet imens analysen kørte.

Erfaring fra denne analyse kan bruges til mange andre analyser. Man kan bruge analysen til at måle tidsforbrug på en hjemmeside!

15) Yderligere referencer

- [1] <https://www.edureka.co/community/53384/spark-dataframe-vs-dataset> (27.04.2020)
- [2] <https://www.linux.com/what-is-linux/> (27.05.2020)
- [3] <https://www.digitalocean.com/community/tutorials/how-to-import-modules-in-python-3> (24.05.2020)
- [4] https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html (27.05.2020)
- [5] https://www.w3schools.com/python/python_intro.asp (27.05.2020)
- [6] <https://www.greelane.com/da/videnskab-tech-math/math/what-is-a-histogram-3126359/> (27.05.2020)
- [7] <https://datavizproject.com/data-type/sunburst-diagram/> (27.05.2020)
- [8] <https://docs.python.org/3/howto/unicode.html> (01.06.2020)
- [9] <https://www.computerhope.com/jargon/u/url.htm> (07.06.2020)
- [10] <https://da.wikipedia.org/wiki/PhpMyAdmin> (11.09.2020)
- [11] [https://da.wikipedia.org/wiki/Docker_\(software\)](https://da.wikipedia.org/wiki/Docker_(software)) (11.09.2020)
- [12] <https://da.wikipedia.org/wiki/MySQL> (12.09.2020)
- [13] <https://docs.djangoproject.com/en/3.1/topics/install/> (11.09.2020)
- [14] <https://gist.github.com/Geoyi/d9fab4f609e9f75941946be45000632b> (14.09.2020)