Fachbereich Technik
Abteilung Elektrotechnik und Informatik

University of Applied Sciences
HOCHSCHULE
EMDEN·LEER

Dokumentation des Semesterprojekts

Entwicklung einer webbasierten Statistikanwendung
zur Auswertung univariater Daten

Prüfer: Prof. Dr. Thomas Preuss
vorgelegt von: Helmut Brünen, Christian Kitte
vorgelegt am: 01. Juli 2018

Inhaltsverzeichnis

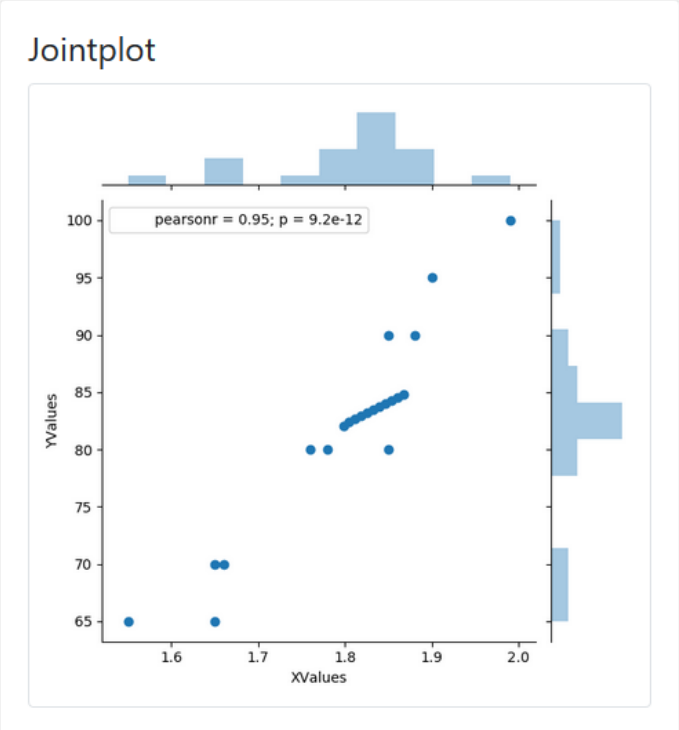
1 Einleitung

Die Anwendung „StatWorker“ ist ein webbasiertes Anwendungsprogramm zur Auswertung von univariaten Daten. Dabei hat der Nutzer die Möglichkeit, eine Datei einzulesen oder beliebig viele Datensätze händisch in eine Tabelle einzugeben. Das Einlesen von Dateien ist als CSV-Datei mit Semikolon als Trennzeichen oder als Excel ab der Version 2007 (.xlsx) möglich.

Auf der Seite „Berechnung“ werden dem Nutzer nach Angabe der Quelldaten durch Drücken des „Berechnen“-Buttons verschiedene Auswertungen in Form von Plots angezeigt. Diese geben Auskunft über die Verteilung der Daten. Es wird bei der Anwendung vorausgesetzt, dass der Nutzer des Programms diese interpretieren und Rückschlüsse auf eine mögliche Verteilung ziehen kann.

Die Berechnung des Joint-Plots bei der Normalverteilung und der Binomialverteilung zeigt zudem noch den Pearson´schen Korrelationskoeffizienten an. Dieser zeigt die Korrelation und damit Abhängigkeit der Variablen an.

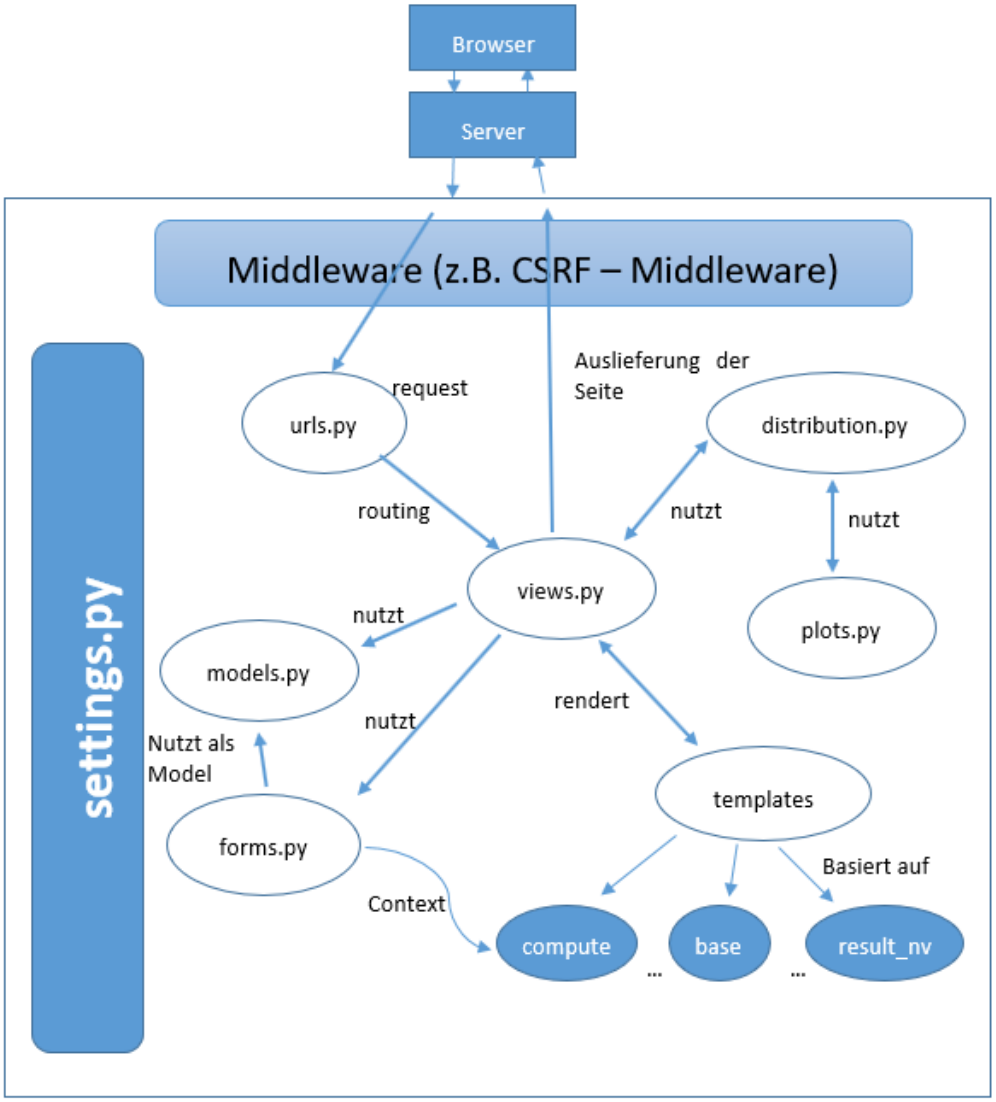
Als Beispiel soll hier die folgende Grafik gezeigt werden. Es handelt sich dabei um den Zusammenhang von Körpergröße (X-Wert) und dem Körpergewicht (Y-Wert):



2 Serverseitiger Aufbau

2.1 Die Architektur

Einer Webanwendung mit Django liegt immer eine eher feste Architektur zugrunde und basiert auf einer Umkehrung des Kontrollflusses. Dieses Prinzip wird auch als Inversion of Control (Umkehrung des Kontrollflusses, Hollywood Prinzip: Don't call us, we'll call you) bezeichnet und bedeutet, das einfachere Module ereignisbezogen spezifischere Module aufrufen. Die folgende Grafik zeigt den Aufbau des Programms:



2.1.1 Die Datenverarbeitung

Grundsätzlich liegen die für die Erstellung der Grafik verwendeten Daten entweder innerhalb einer Exceldatei, eines CSV-Datei oder aber in Form zweier Stringarrays vor. Dies ist für die weitere Verarbeitung wenig sinnvoll, da die verschiedenen Formate berücksichtigt werden müssten.

Aus diesem Grunde findet eine Vorverarbeitung statt. Hierfür ist die Methode prepare_file und __create_csv_file der Klasse ComputeRequestModel zuständig. Sie erzeugt aus den Dateien eine CSV-Datei mit Komma als Trennzeichen und Punkt als Dezimaltrennzeichen. Im Bedarfsfall könnte sie auch für weitere Dateiarten erweitert werden. Für den Zugriff auf die Exceldatei wird das Paket OpenPyXL verwendet.

Nach der Erstellung der Datei wird diese im Media-Verzeichnis zwischen gepeichert und die Ursprungsdatei gelöscht. Um Namenskonflikte zu vermeiden, wird sie mit einer GUID zu einem eindeutigen Namen umbenannt. Dieser wird in Folge an die Klasse Distribution übergeben.

2.2 stat_worker_site - Die .py-Programme und deren Aufbau

Auch wenn durch die Namensgebung der Eindruck vermittelt wird, gehört dies Verzeichnis eher zur Infrastruktur von Django als zu der hier vorgestellten Anwendung. Nach einer Installation von Django ist dies die erste erstellte Struktur, stellt aber noch keine lauffähige Webanwendung dar. Weiter erhält man durch die Installation des Frameworks einen Satz von ausführbaren Werkzeugen.

So kann mit dem Befehl ***django-admin.py startapp appname [destination]*** eine neue Anwendung erstellt werden. Aus der Sicht von Django bedeutet dies, dass eine vorgegebene Verzeichnisstruktur wie die in 2.3 näher ausgeführte erzeugt wird und deren Vorhandensein notiert wird. Hierbei ist Django so effizient, dass direkt nach der Erstellung bereits eine Default-Seite aufrufbar ist.

Weitere nützliche Befehle sind ***manage.py makemigrations, manage.py runserver [ip:port]*** oder ***manage.py createsuperuser.***

2.2.1 Die __init__.py

Diese Datei gibt einen Hinweis darauf, dass Verzeichnis als Ganzes als ein Python-Paket betrachtet wird. Da hier keine Anforderungen existieren, ist sie leer.

2.2.2 Die „settings.py“

Setting.py ist die zentrale Konfigurationsdatei von Django und enthält alle notwendigen Einstellungen für den Aufbau und Laufzeit der aufgesetzten Apps. Im Wording von Django bezeichnet eine

App eine in sich geschlossene Einheit. Eine Website kann hier durchaus aus mehreren Apps bestehen.

Zu den wichtigsten Einstellungen zählen:

- BASE_DIR – Angabe des Basisverzeichnisses
- SECRET_KEY – Ein (geheimer) Schlüssel zur Verschlüsselung
- DEBUG – Regelt die Ausgabe von Fehlermeldungen
- ALLOWED_HOSTS – Muss im Livebetrieb mit mind. der eigenen IP gefüllt sein
- INSTALLED_APPS – Eine Liste der vorhandenen Apps
- MIDDLEWARE – Eine Liste der eingesetzten Middleware
- ROOT_URLCONF – Legt das Modul zur Auswertung der Requests fest (hier urls.py)
- TEMPLATES – Verzeichnis der Templates
- WSGI_APPLICATIONS – Verweist auf die Konfigurationsdatei der Anwendung
- DATABASE – Verweist auf Datenbanktyp und Name der verwendeten Datenbank
- AUTH_PASSWORD_VALIDATORS – Eine Liste der eingestzeten Validatoren
- STATIC_URL – Definiert die Route zu den statischen Inhalten
- MEDIA_URL – Definiert die Route zu den Medieninhalten
- MEDIA_ROOT – Definiert das Verzeichnis zu den Medieninhalten
- SESSION_ENGINE – Definiert den einzusetzenden Mechanismus (hier Cookies)

2.2.3 Die urls.py

Die Datei wird in der settings.py definiert (der Name könnte also grundsätzlich auch anders sein) und bildet den zentralen Einstieg aller eintreffenden Requests. Grundsätzlich lassen sich hier somit alle Anfragen weiterleiten. Praxis ist jedoch, die Weiterverarbeitung an ein Modul der Anwendung weiterzureichen in der Form, dass das entsprechend Modul included wird. Die urls.py gibt in unserem Fall einen Verweis auf die urls.py der App „stat_worker_app“.

```
urlpatterns = [
    # path('admin/', admin.site.urls),
    path('', include('stat_worker_app.urls'))  # redirect to the app's route handler
]
```

2.2.4 Die „wsgi.py“

Die „wsgi.py“ ist die initiale Einstellungsdatei für die Umgebung der Anwendung und zeigt auf, wo die Einstellungen zur Anwendung zu finden sind. Diese Information wird in der Umgebungsvariable ***DJANGO_SETTINGS_MODULE*** gehalten und bei Instanziierung der Anwendung mit ***get_wsgi_application*** verwendet.

```
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "stat_worker_site.settings")

application = get_wsgi_application()
```

WSGI steht für ***Web Server Gateway Interface*** und ist eine Schnittstellenspezifikation für die Sprache Python zwischen Webserver und Webframework bzw. Web-Application Server und so eine Trennung und späteren Tausch ermöglichen. Somit ist es auch eine Middleware.

2.3 Stat_worker_app – Die Module und deren Aufbau

Dieses Verzeichnis „stat_worker_app“ entspricht der App, die realisiert wurde. In diesem Verzeichnis, welches mit dem Befehl „startapp Appname“ erzeugt werden kann, liegen die gesamten py-Anwendungsdateien.

2.3.1 Die __init__.py

Diese Datei gibt einen Hinweis darauf, dass Verzeichnis als Ganzes als ein Python-Paket betrachtet wird. Da hier keine Anforderungen existieren, ist sie leer.

2.3.2 Die „apps.py“

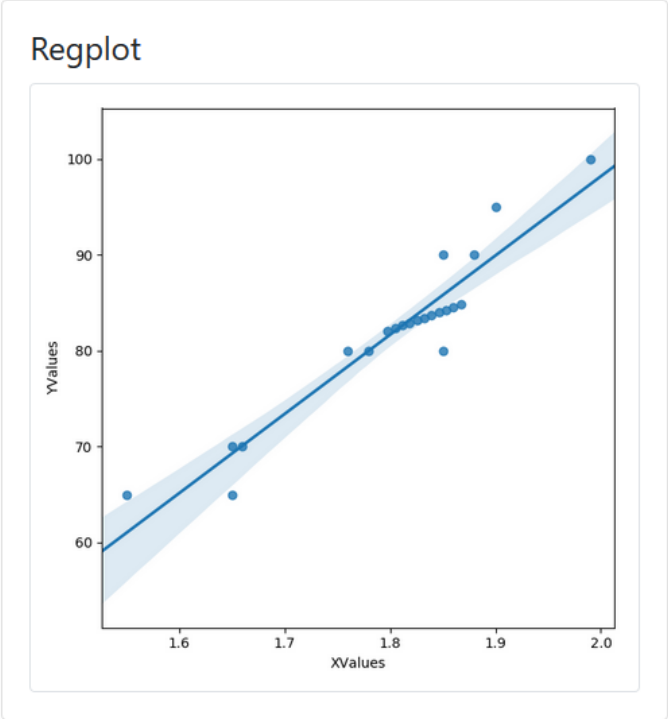
Die apps.py beinhaltet nur eine kleine Klasse StatworkerConfig und erbt von AppConfig. Hier wird der Anwendung mitgeteilt, wie der Name der App lautet.

2.3.3 Die „distribution.py“

Die distribution.py ist die zentrale Berechnungs- und Plot-Datei. In dieser py-Datei wird festgelegt, welche Plots für die Normalverteilung, der Gleichverteilung und der Binomialverteilung herangezogen werden. Ebenfalls erfolgt hier die Berechnung der Grafiken. Zur Erzeugung dieser wird auf die Bibliothek „Seaborn“ zurückgegriffen.

Es werden die folgenden Plots pro gewählter Verteilung erzeugt (mit dem Regplot als Beispiel):

1) Normalverteilung:
    a. Box-Plot (Box-Whisker-Plot = Streuungsdiagramm)
    b. Regplot (Regressionsgerade)



    c. Pairplot (Aufbereitung von x und y-Werten)
    d. Jointplot (mit Korrelationskoeffizienten und Fehlerwahrscheinlichkeit)
    e. Residuenplot (vertikaler Abstand der y-Ausprägungen zur Regressionsgeraden)
    f. Histoplot (Histogramm der y-Daten)
    g. Distplot (Verteilungsfunktion/Dichtefunktion)
2) Gleichverteilung:
    a. Box-Plot (wie oben)
    b. Pairplot (wie oben)
    c. Histoplot
3) Binomialverteilung:
    a. Box-Plot (wie oben)
    b. Jointplot (wie oben, da Binomialverteilung sich der Normalverteilung annähert)
    c. Pairplot (wie oben)
    d. Distplot (wie oben)

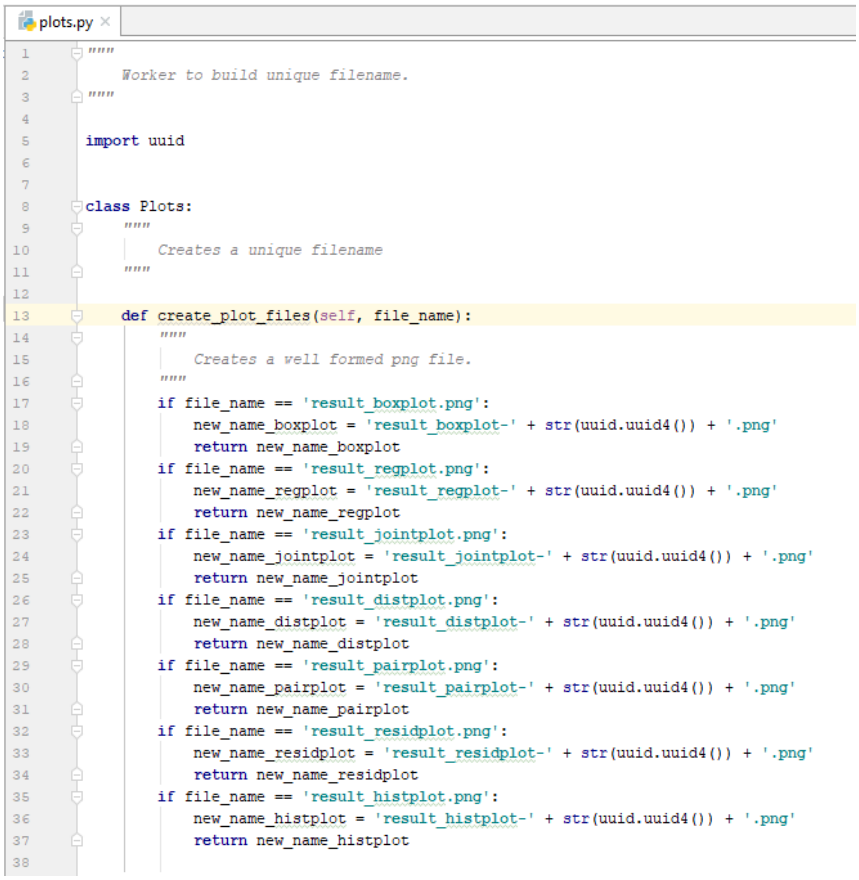
Der Rückgabewert dieser Funktion ist ein Array, dass die für die Session eindeutigen Namen der erzeugten Dateien enthält. Somit existiert für jeden Aufruf ein Satz eindeutig benamter Dateien. Nach der Erzeugung der Grafikdateien wird die ursprüngliche Datendatei gelöscht.

2.3.4 Die „forms.py“

Die forms.py enthält das Model für das Formular zur Eingabe der Datei bzw. der Daten und die verwendeten Felder method (Normalverteilung, Gleichverteilung oder Binomialverteilung), source (Datei oder freie Eingabe), file (Datei), data1 und data2 als die Spalten 1 und 2 bei freier Eingabe.

2.3.5 Die „plots.py“

In der plots.py wird aus einem übergebenen Namen ein eindeutiger Name erstellt und zurückgegeben. Hierfür wird mittels des Python-Pakets „uuid“ eine GUID generiert, angehangen und zurückgegeben.



2.3.6 Die „models.py“

Die models.py als zentrale Datei verfügt über die Klasse ComputeRequestModel(models.Model) und kapselt eine konkrete Anforderung für eine Berechnung. Innerhalb der Klasse befinden sich Funktionen zur Überprüfung von validen Tabellen bzw. Dateien und die ebenfalls wichtige Funktion zur Generierung der CSV-Dateien. Exemplarisch soll hier nur die Funktion zur Erstellung der .csv-Datei dargestellt werden:

```python
def __create_csv_file(self, data):
    """
        Creates a well formed csv file (',' separated with decimal '.' instead of ',') within media path and
        returns its full qualified name.
    """

    path = join(settings.MEDIA_ROOT)
    name = 'data-' + str(uuid.uuid4()) + '.csv'

    file_path = join(path, name)
    new_file = open(file_path, 'w', newline='')

    csv_writer = csv.writer(new_file)
    for item in data:
        csv_writer.writerow(item)

    new_file.close()

    return new_file.name
```

2.3.7 Die „urls.py“

Die urls.py der Anwendung parst die eingehenden Routen und verweist auf die Views. Durch die Interaktionen der Nutzer mit dem Programm holt sich die View so je nach gewähltem Menüpunkt die entsprechende html-Datei, z.B. „about.html“ für „Über uns“.

urls.py

```python
"""
    The app's route handler.

    Handles requests an redirect them to an appropriate view.
"""

from django.urls import path
from . import views

urlpatterns = [
    path('', views.base, name='base'),
    path('about', views.about, name='about'),
    path('compute', views.compute, name='compute'),
    path('imprint', views.imprint, name='imprint'),
    path('methods', views.methods, name='methods'),
    path('privacy', views.privacy, name='privacy'),
    path('results_nv', views.results_nv, name='results_nv'),
    path('results_gv', views.results_gv, name='results_gv'),
    path('results_bv', views.results_bv, name='results_bv'),
    path('welcome', views.welcome, name='welcome')
]
```

2.3.8 Die „views.py“

Die views.py enthält Methoden zur Verarbeitung der eingehenden Requests. Diese können einfach nach ihrer Methode (POST, GET, …) unterschieden und abgearbeitet werden. Im einfachsten Fall erfolgt ein einfaches Rendern einer Seite als Rückgabe.

views.py

```python
"""
    Handels the requests.

    Most of them simply returns an template based page
"""

from django.core.files.storage import FileSystemStorage
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .forms import ComputeRequestForm
from .models import ComputeRequestModel
from .distribution import Distribution


def base(req):
    """
        Returns the base

        :param req: the request
        :return: base.html
    """
    return render(req, 'app/base.html')


def about(req):
    """
        Returns the about

        :param req: the request
        :return: about.html
    """
    return render(req, 'app/about.html')


def compute(req):
    """
        For GET: returns the compute
        For POST: Evaluate the data and redirect to an result page

        :param req: the request
        :return: compute.html
    """
    if req.method == 'POST':
        global list_pictures

        model = ComputeRequestModel()
        form = ComputeRequestForm(req.POST, instance=model)  # bound form

        if req.POST['source'] == '1':  # file
            file = req.FILES['file']  # raise an error if missing

```

2.4 Die Verzeichnisse „static“, „media“ und „templates“

In den Verzeichnissen „static“ und „templates“ liegen statische Dateien. Die folgenden beiden Abschnitte beschreiben die Inhalte.

2.4.1 Das Verzeichnis „static“

Das Unterverzeichnis static enthält in den Unterverzeichnissen „img“ und „app“ zum einen die in der Anwendung verwendeten statische Bilder, und zum anderen enthält es die Javascript-Datei „clientside.js“. Diese Datei dient der Steuerung der Anwendung auf der Clientseite.

Als weiteres werden hier die zur Ausgabe erzeugten Bilder der Plots als PNG-Bilder gespeichert, um sie als statische Ausgabe in den entsprechenden Ausgaben verlinken zu können.

2.4.2 Das Verzeichnis „media“

Das Verzeichnis dient dem Halten der hochgeladenen Dateien und der temporären Speicherung von bearbeiteten Dateien.

2.4.3 Das Verzeichnis „templates“

In diesem Verzeichnis sind die templates zum Rendern der HTML-Inhalte der Anwendung abgelegt. So enthält dieses Verzeichnis folgende html-Dateien:

1) about.html (Über uns)
2) base.html (Startbildschirm)
3) compute.html (Menüpunkt „Berechnungen“)
4) error.html (Fehlermeldung)
5) imprint.html (Impressum)
6) methods.html (Menüpunkt „verwendete Methode“)
7) privacy.html (Datenschutz)
8) result_bv.html (Ergebnisseite Binomialverteilung)
9) results_gv.html (Ergebnisseite Gleichverteilung)
10) results_nv.html (Ergebnisseite Normalverteilung)
11) welcome.html (Begrüßungsbildschirm)

Dieses Verzeichnis wird durch eine Einstellung in der settings.py erreicht und angesprochen:

settings.py

```
53  # some definition for template engineering
54  TEMPLATES = [
55      {
56          'BACKEND': 'django.template.backends.django.DjangoTemplates',
57          'DIRS': [os.path.join(BASE_DIR, 'stat_worker_app/templates')]
58          ,
59          'APP_DIRS': True,
60          'OPTIONS': {
61              'context_processors': [
62                  'django.template.context_processors.debug',
63                  'django.template.context_processors.request',
64                  'django.contrib.auth.context_processors.auth',
65                  'django.contrib.messages.context_processors.messages',
66                  'django.template.context_processors.media',
67              ],
68          },
69      },
70  ]
```

3 Clientseitiger Aufbau

3.1 Grundsätzliche Architektur

Clientseitig ist die Anwendung als eine Single Page Application (SPA) aufgebaut. Lediglich die Hauptseite wird zu Anfang geladen. Diese beinhaltet alle Ressourcen für die weitere Ausführung wie Navigationselemente oder Skripte. Während der Ausführung wird im folgenden benötigter Content dynamisch nachgeladen, respektive Anforderungen und Daten hochgeladen.

3.2 Das Menü/Navigation

Die Anwendung besteht aus sechs unterschiedlichen Seiten:

1) Willkommen – Hier erfolgt eine kurze Einführung und Abholung des Nutzers.
2) Berechnung – Hier hat ein Nutzer die Möglichkeit, Daten als CSV-Datei (Semikolon als Trennzeichen) oder als Excel 2007 (.xlsx) hochzuladen oder aber manuell in eine Tabelle einzugeben. Hierzu wählt er eine der angegebenen Optionen ***Daten als Datei hochladen (.csv oder .xlsx)*** oder aber ***Daten manuell eingeben*** aus.

   Wird die manuelle Eingabe gewählt, so wird eine Tabelle eingeblendet, in der die Daten direkt eingegeben werden können. Über die nun sichtbaren grünen Buttons können Zeilen hinzugefügt, oder aber entfernt werden.

   Mit den Buttons ***Berechnen*** bzw. ***löschen…*** wird die Berechnung angestoßen oder aber die gemachten Eingaben verworfen.

   Das Ergebnis selbst in Form einer Auswahl von statistischen Diagrammen wird durch die Selektion einer der im Dropdown-Menü aufgeführten Methoden bestimmt (siehe 2.3.3).
3) Verwendete Methoden – Auf dieser Seite erfolgt eine kurze Einführung zu den in der Anwendung berücksichtigten Methoden Normal-, Gleich- und Binominalverteilung.
4) Über uns… – Hier wird kurz über die Intention der Seite sowie den umgebenden Rahmen eingegangen.
5) Impressum – Ein Impressum mit den auch für private Seiten notwendigen Angaben.
6) Datenschutz – Eine Erklärung zum Datenschutz, in der hauptsächlich klargestellt wird, dass die eingegebenen Daten nicht längerfristig gespeichert werden.

3.3 Verwendete Bibliotheken

Für die Dynamik und die Gestaltung der Seite wurden neben eigenem Code das CSS Framework ***Bootstrap 4***, ***JQuery 3*** sowie die Hilfsbibliothek ***Popper*** verwendet.

3.3.1 Clientside.js

Diese Datei enthält Javascriptcode für die Dynamik der Seite. Die Dynamik auf der Cleintseite zu machen bietet sich an, da so Zeit und Daten gespart warden können. Würde dies nicht serverseitig gemacht, müsste für jedes kliene Detail ein Roundtrip zum Server erfolgen.

Die Funktionen ***getContentFor*** bzw. ***postContentFor*** handelt mittels Fetch API alle anfallenden Requests. Die sind zum einen Anforderungen für neuen Content und zum anderen der Upload der für eine Berechnung benötigten Daten.

Die Funktion ***getPage*** handelt alle Menüereignisse und sorgt für die Anforderung der notwendigen Inhalte.

CatchFormCompute fängt das Submit des Formulars ab und kontrolliert die Daten. An dieser Stelle werden auch die manuell eingegebenen Daten kontrolliert und für den Versand bearbeitet. Am Ende steht ein FormData Object, welches mittels Post zum Server transferiert wird.

Die verbleibenden Funktionen sind lediglich Hilfsfunktionen z.B. zur Erweiterung oder Verkleinerung der Tabelle.

4 Zusammenfassung

Mit der vorliegenden Webanwendung konnten wir zeigen, dass sich mit Hilfe von Django bereits mit relativ wenig Aufwand eine sehr schöne Anwendung erstellen lässt. Die auf Clientseite notwendige Programmierung ist nicht Django geschuldet, sondern der Vermeidung von unnötigen Roundtrips zum Server und auch bei anderen Frameworks und Sprachen eher sinnvoll.

Serverseitig konnten mit der Struktur und der Infrastruktur von Django alle Aufgaben wie Routing, Templating, Uploads und Auslieferung von Seiten elegant umgesetzt werden und die durch die Aufgabenstellung zu lösenden Probleme gelöst werden.

Es ist zwar nicht immer die aus Sicht von Django optimale Lösung, aber im Verlauf der Umsetzung konnten wir uns sehr viel an Hintergrundwissen aneignen und so eine gute Basis für eine weitere Einarbeitung aneignen.

Als ein möglicher Punkt muss hierbei die serverseitige Validierung und Gestaltung von Formularen mithilfe der Infrastruktur von Django gesehen werden. Hier zeigten sich bei der Programmierung Probleme, die zu Gunsten des ganzen Projektes anders gelöst wurden, als es im Sinne von Django ideal wäre.

Trotz dessen bleibt ein kritischer, aber auch positiver Eindruck des Frameworks. Dies vor allem, wenn man die Vorteile der Sprache Python einfach online verfügbar haben möchte.