Accident Number Forecasting

Introduction (Mission 1)

Forecasting the accidents counts is an important topic to prepare aids, ambulances for victims. Also, it helps to expect the hospitals preparations. Also, it helps to avoid it. This dataset has been prepared, cleaned, analyzed, and visualized as all in the All_Data_Analysis_Visualization.ipynb. and in from this Analysis, I found the following properties:

- 1764 rows of data.
- This data is from 2000 to 2020
- It has three types of accidents as follows 'insgesamt', 'Verletzte und Getötete', and 'mit P ersonenschäden'
- These accidents have the following categories 'Alkoholunfälle', 'Fluchtunfälle', and 'Verke hrsunfälle

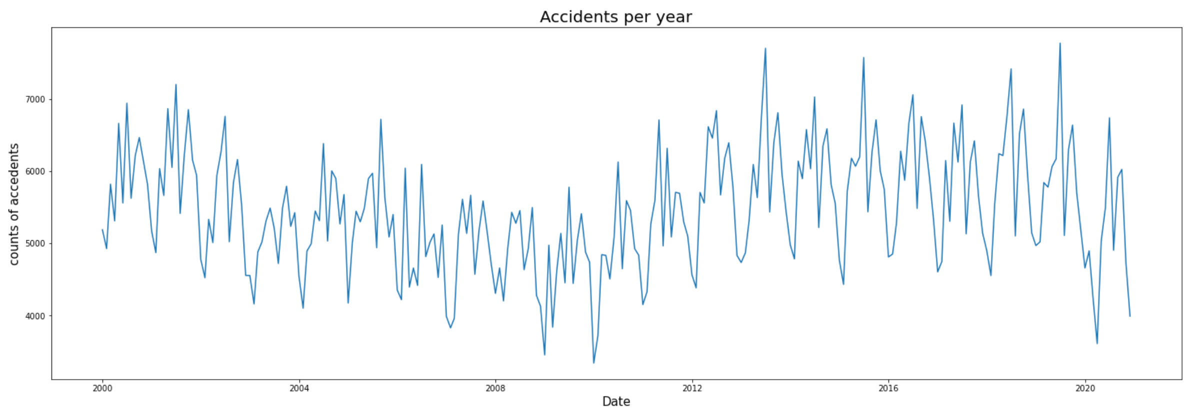
Analyzing all data Notebook Parts (All_Accidents_Data_Analysis.ipynb)

In the cleaning part, I applied the following to clean it before analysis or visualization.

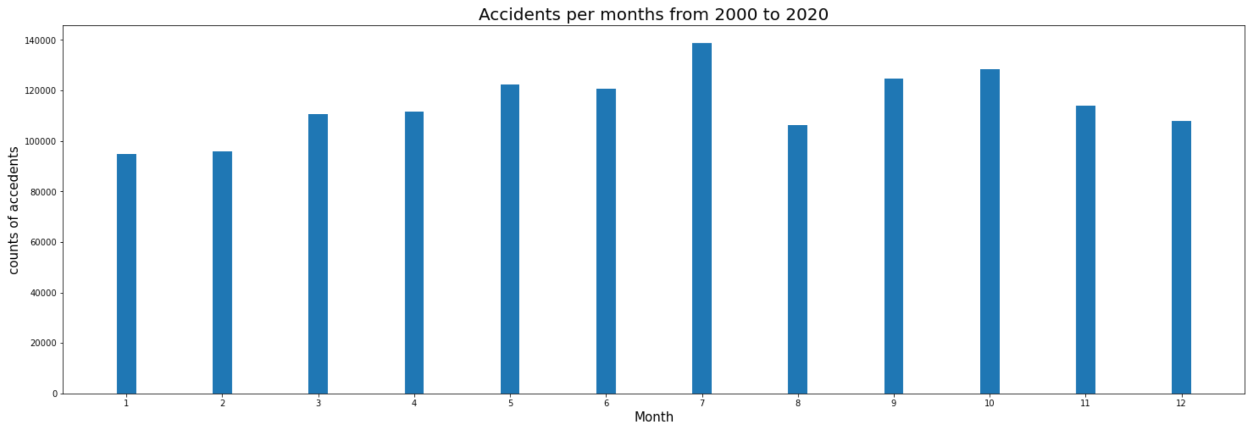
- Remove the row with month equal=Summe.
- Create the date column.
- Sort the values according to the date column. This step is mandatory in time-series analysis.
- Put the date column as the main index.
- Fill the month column with values from 1 to 12

In the Analysis data part, I analyzed and visualized the data to explore some insights and relationsip between data columns such as,

- Accidents per year from different types and categories.

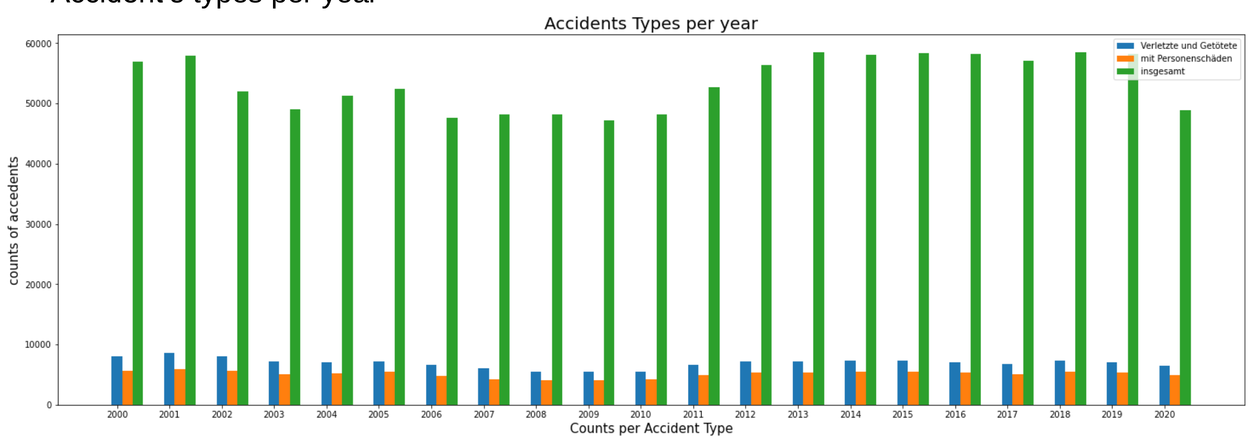


- Summation Accidents per month



We notice from above figure that July month has the maximum number of accidents.

- Accident's types per year



From here, we found that 'insgesamt' caused the highest count of accidents.

- Obtaining the relationship between accidents' types and categories. We notice from her that the 'Verkehrsunfälle ' has the highest number of accidents.

```
category          accident_type
Alkoholunfälle    Verletzte und Getötete      5216
                  insgesamt                  11026
Fluchtunfälle     Verletzte und Getötete     11312
                  insgesamt                 221616
Verkehrsunfälle   Verletzte und Getötete    128906
                  insgesamt                 891374
                  mit Personenschäden       106986
```

In Saving data part

After all cleaning steps, the data is save to be used in the Insgesamt_Accidents_Analysis_and_Modeling notebook to build the model.

Insgesamt Accidents Analysis and Modeling Notebook

Insgesamt_Accidents_Analysis_and_Modeling)

In this notebook, I will model the data of only 'insgesamt' from 'Alkoholunfälle' category. The data in our hand has some types of accidents, and each type has some categories. So, each type with each category should be modelled individually. To model all the data types and categories, we should have 7 models.

Since, the required values in the example has the 'insgesamt' type and the Alkoholunfälle category, I extracted only their related data from the dataset. In prediction time, we should call the most suitable model according to the data category. Let's go through notebook parts.

Data Preparing

Where the cleaned data is read and extract the data of 'Alkoholunfälle' category and 'insgesamt' accident type.

Preprocessing

- Splitting Data into training and testing.
- Data Normalization.
- Preparing LSTM subsequences

Modelling

As mentioned above, I modelled the 'insgesamt' type from the Alkoholunfälle category.

Although, all rows in my hand are 252 only. I could tune the LSTM to model it.

The model architecture is as follows:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rnn (RNN)                    (None, 64)                17408
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 17,473
Trainable params: 17,473
Non-trainable params: 0
_________________________________________________________________
```

During modeling, I used the following configuration parameters:

- Epoch=10000 with early stoping, learning rate= 0.00001, LSTM size =64, batch_size=4, validation split = 0.25 of training data, testing split = 0.1 of data, loss is mean square error, patience parameter in early stopping is 2, and dense layer size=1.
- keras is used to build, train this model.

Note: the model is trained and save in Insgesamt_Accidents_Analysis_and_Modeling.ipynb.

Inference (Inference.ipynb)

I tried to forecast the value of the following example (inside the challenge):

Category: 'Alkoholunfälle'
Type: 'insgesamt
Year: '2021'
Month: '01'

The predicted value from the 1st model is 15, the real value is 35
Note that the inference python code is in Inference the notebook.

Local Deployment using FastApi (app.py)

To do model serving, it is required to deploy the model to the server. In this scenario, put the model on the server "End point" and this point accepts http request, so we need a rest API, inside it.

I used the FastApi to deploy the model. The app.py includes all the code of FastApi as follows:

- Imports
- Creating API
- Preprocessing part which looks like the preprocessing in the training part.
- Loading the model locally if it is available or from the server if it is deployed on the server.
- The get request root which returns the message to the user.
- The post request which receives Json containing the values of year and month and returns the value of the prediction.
- Necessary definitions and their validators.

1. Running the Get request, use the following command from gitbash:

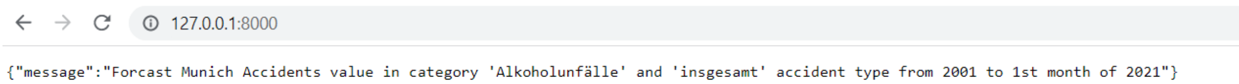
```
uvicorn app:app –reload
```

Use the following URL on any browser http://127.0.0.1:8000/

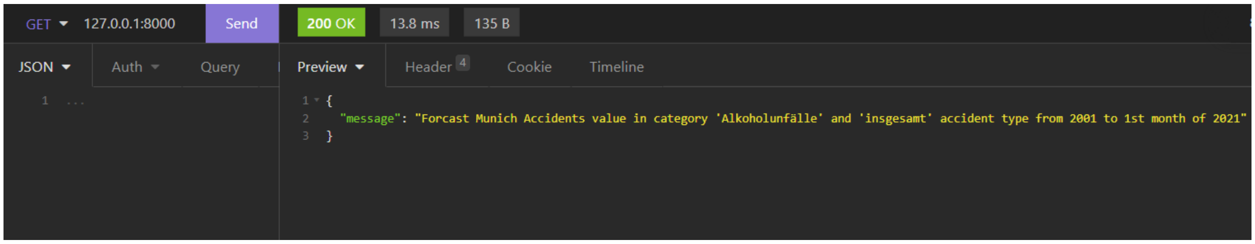
or

The Get request using Insomina on http://127.0.0.1:8000/

The output looks like the following:



Or



2. Running the Post to pass a certain value in JSON file, then

I tried to send the fill the post request with json using "Ensomnia" application.

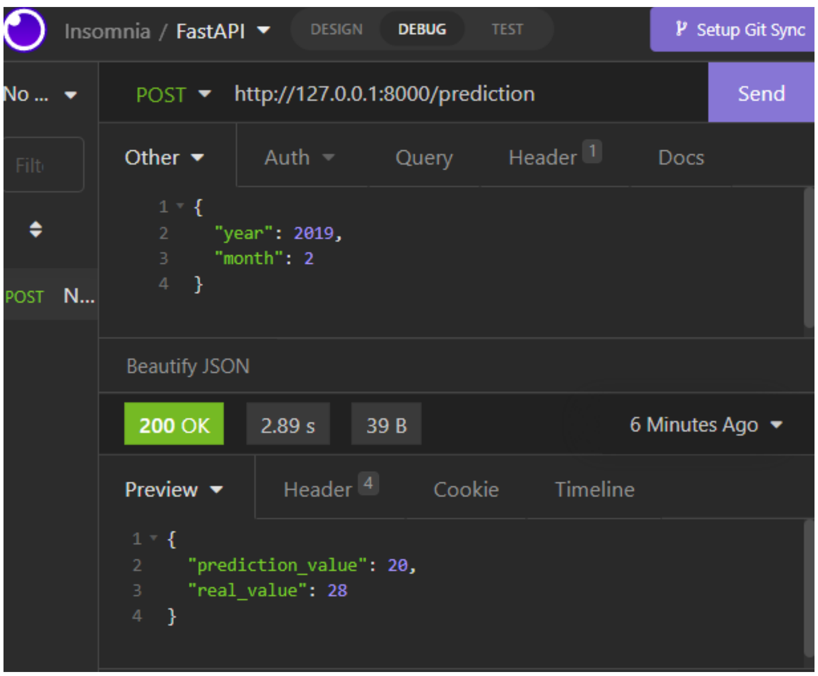


Figure 1 shows the post request input and return values

Google Cloud Deployment

I deployed the best_model.h5 to google cloud forecast123_bucket bucket inside project ForecastingAccidentsApp. The model link is at:

```
'gs://forecast123/best_model.h5'
```

Containerization

Containerization is essential in the development cycle because it enables you to reproduce your colleague's work regardless of the OS used to develop the software component without sophisticated steps. Also, it simplifies the deployment and the automation on the server.

One of the most famous tools used for this purpose is the Docker. I used it in this project to containerize my app.py, gap.py which used FASTAPI to predict the count of accidents local deployment and google cloud deployment application files.

To do that I did the following:

- Writing the Dockerfile.
- Building Docker image using the following command:

  `$ docker build -t c_app:1.0 .`

- Running the container

  $ docker run -d --name contain -p 8000:5000 -t -i c_app:1.0

- Entering inside the container its bash run the following:

  $ docker exec -it container_id /bin/bash

  container_id: output from previous step.

  $ uvicorn app:app --reload

  $ uvicorn gapp:app --reload

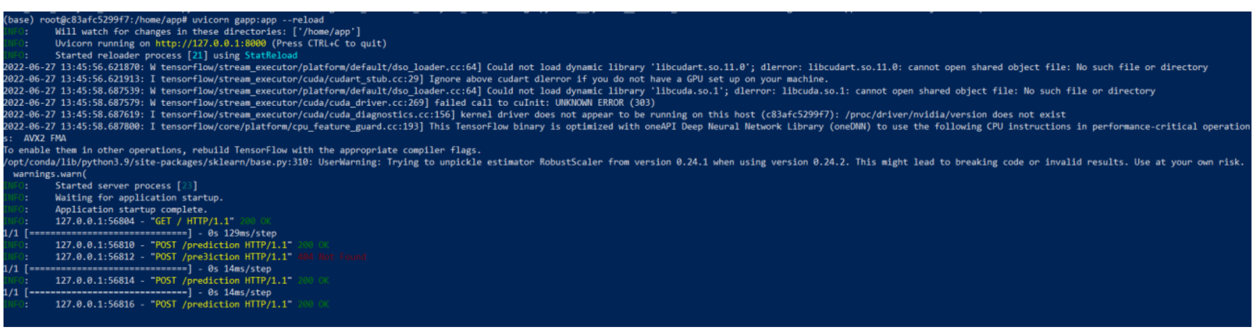


Figure 2: Output after Executing the above Commands.

- Testing the page contents of the page the get request to see the index page message and the post request using the following commands:
  curl -X POST -H "content-Type: application/json" -d'{"year":2020, "month":3}' 127.0.0.1:8000/prediction

```
(base) root@a2ff9aacc10a:/home/app# curl  http://127.0.0.1:8000
{"message":"Forcast Munich Accidents value in category 'Alkoholunfälle' and 'insgesamt' accident type in 2020 and 12st month of 2021"}
```

```
(base) root@a2ff9aacc10a:/home/app# curl -X POST -H "Content-Type: application/json" -d'{"year":2020, "month":3}'  127.0.0.1:8000/prediction
{"prediction_value":23}(base) root@a2ff9aacc10a:/home/app#
```

It works fine in both get and post as shown in the above figures.