CMP6230 – COURSEWORK

Module Code: CMP6230
Module Title: Data Management and Machine Learning Operations
Coordinator Name: Emmett Cooper

Student Name: Dharmarlou Bowen
Student ID: 20123076
Word Count: 4122

Date: 20.10.2022

Table of Contents

Table of Figures

Data Comparison

In this section an appropriate dataset will be acquired for use of developing and designing a data management strategy solving a supervised learning task within a specific domain to produce analytical insights.

Dataset 1: Heart Disease Predication

The first data set is concerned with the predication of heart disease. It was acquired from Kaggle, the author of the dataset states that it was created by combining different datasets that were already publicly available. 5 heart datasets that shared 11 common features is what allows them to be combined with them differing on the fact that each stem from different geographical locations. With this dataset's goal being the predication of heart disease this meets the criteria of presenting a classification problem within the realm of supervised learning.

Heart Failure Prediction
Age (integer)
Sex (string)
ChestPainType (string)
RestingBP (integer)
Cholesterol (integer)
FastingBS (integer)
RestingECG (string)
MaxHR (integer)
ExerciseAngina (other/boolean)
Oldpeak (decimal)
ST_Slope (string)
HeartDisease (integer)

Figure 1: Diagram of Heart Disease Dataset

The overall structure of the dataset is comprised of 12 features broken into (**Figure 1**):

- 6 Integer
- 4 String
- 1 Decimal
- 1 Other

The dataset is stored as a CSV file so its implementation to a model should be straight forwards given its been pre-processed. In this dataset the target variable would be 'HeartDisease' as it indicates if the patient suffers from it or not. I have not used this dataset in previous project so it may present some intriguing insights into what factors may be contributing towards heart disease more.

Dataset 2: Stroke Prediction

The second dataset revolves around the goal of stroke predication. It was acquired from Kaggle, the author of the dataset states that the original source for the data is confidential and should only be used for educational purposes. The dataset is comprised of 12 features and 5110 observations.

Each one representing a patient concerning factors such as gender, bmi and if they suffered from a stroke or not as the core feature. All the features present in the dataset (with the exception of each patient's unique identifier) will be used to determine what could correlate to the occurrence of a stroke (**Figure 2**).

Stroke Predication
id (integer)
age (decimal)
gender (string)
hypertension (integer)
heart_disease (integer)
ever_married (other/boolean)
work_type (string)
Residence_type (string)
avg_glucose_level (decimal)
bmi (decimal)
smoking_status (string)
stroke (integer)

Figure 2: Diagram of Stroke Dataset

The predicative problem tackled is a classification problem concerning stroke predication. By analysing the records in the in the dataset we will be attempting to uncover if certain features may serve as more prominent factors in occurrence of strokes. Such features would reveal if a patient may suffer from a stroke or not.

The variable that will be used as the target is the 'Stroke' feature for the establishing of ground truth. This variable is a numerical value that is presented as either a 0 or 1 in the dataset, a 0 means a patient hasn't suffered from a stroke while 1 means the opposite. The dataset is stored in a CSV file.

Dataset 3: Diabetes Predication

The final dataset option is centred around the predication of diabetes. The dataset was acquired from Kaggle, the author of the dataset states that the data originates from the National Institute of Diabetes, Digestive and Kidney Diseases.

The goal of the dataset is to diagnostically predict if a patient may have diabetes or not. The dataset is all comprised of females above the age of 21, the reason for this may be due to the selection of the dataset's instances from a larger dataset.

Diabetes
Pregnancies (integer)
Glucose (integer)
Blood Pressure (integer)
SkinThickness (integer)
Insulin (integer)
BMI (decimal)
DiabetesPedigreeFunction (decimal)
Age (integer)
Outcome (integer)

Figure 3: Diagram of Diabetes Dataset

The overall structure of the dataset is comprised of 9 features broken into (**Figure 3**):

- 7 Integer
- 2 Decimal

The dataset is stored as a CSV file so its implementation to a model should be straight forwards given its been pre-processed. In this dataset the target variable would be 'Outcome' as it indicates if the patient suffers from it or not. I have not used this dataset in previous project so it may present some intriguing insights into what factors may be contributing towards diabetes more.

Chosen Dataset

The dataset that will be used is the Stroke dataset as I am the most familiar with it after using it in previous projects and feel that revisiting it with a different approach may lead to more insightful observations compensating for possible things that may have been missed during its previous use.

Designing and Blueprinting of Pipeline

Now that an appropriate dataset has been acquired we can begin the process of designing and planning our data pipeline, this section is broken into 5 sub-sections concerning data ingestion, pre-processing, model development and deployment in addition to the monitoring of said model. The who, what, how and why will be answered throughout each section.

Data ingestion

Data ingestion is the first core element of data analytics pipeline as pipelines are redundant without any data. It is responsible for the process of steps that result in the transmission of data from a variety of sources to an appropriate medium such as data lakes and warehouses.

This stage is important due to the fact that it is the stating point of the data pipeline resulting in to playing a great part in the scalability and development of the entire pipeline. Such factors are influenced by the structure of the data and its method of storage due to each database engine offering different benefits and drawback to the performance of a given task. The two data processing systems that are generally utilized in data science are OLAP (Online Analytical Processing) and OLTP (Online Transactional Processing). For which one is chosen to be used to directly dependent on the tasks it is aimed to accomplish.

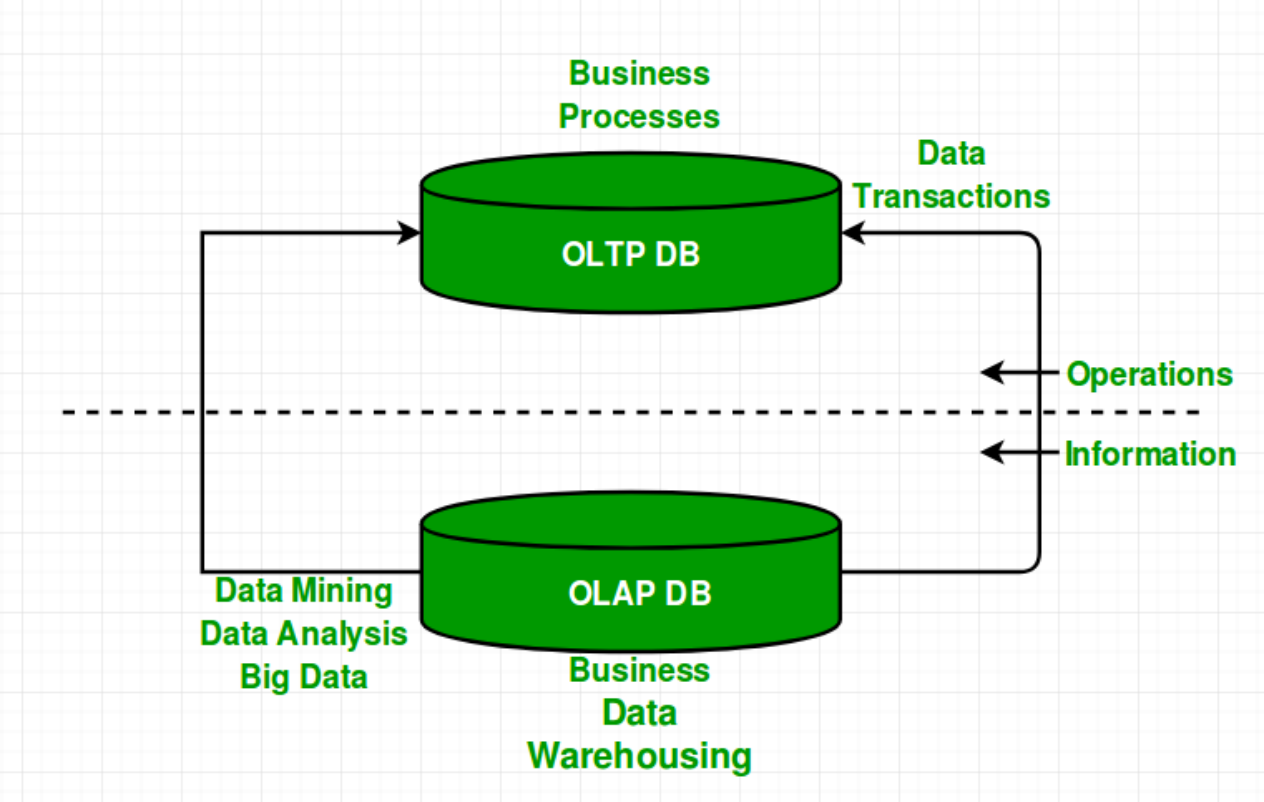


Figure 4: Diagram of OLAP and OLTP Systems

It should also be stated that taking into consideration how the data is dealt with in correspondence with ethical limitations, especially when a system may be processing sensitive data such as personal information the security and privacy of said data is paramount.

The transformation process of data is this stage would be performed by a mix of data scientists and engineers as the output would be used in the pre-processing stage. The ingestion of data begins with the extraction of data from a given source. In the case of the data used in this report it originates from a CSV sourced from Kaggle. This process of extraction and ingestion will be making use of tools such as airflow, redis and docker in junction with python on a ubuntu operating system. In addition to all of this using Great Expectation would aid in the validation of the data.

Once appropriate ingestion has been performed on the data the output would result in the data being easily accessible, but its important to keep in mind that particular processes in this stage can typically be done in the pre-processing stage such as data validation, checking and parsing.

Data Pre-processing

Once the dataset has been ingested and stored it would be ready to be outputted to the pre-processing stage of the pipeline. This stage places emphasis on the cleaning, preparing and transforming of data for further analysis, which entails if required:

- removing or imputing of missing values
- dealing with outliers
- removing irrelevant data
- encoding categorical values
- oversampling appropriate values

The key tool that would allow these tasks to be achieved would be the use of the pandas library as it possesses the appropriate methods to meet all these tasks. Each task may be performed to varying degrees depending on the type of data being dealt with in addition to the goal of the pipeline. The handling of missing data can effect the outcome of analysis performed on said data. Two methods are used to do this, imputation being the first where the missing values are replaced with appropriate values to take their place: The alternative being the simple removing of data.

Data encoding serves the purpose of ensuring that all features can be understood by the models there going to be implemented into by encoding them into numerical values. This process is performed on categorical values due to model not being able to understand such categorical values. Method to solve this involve hot encoding. Outliers can deeply affect the outcome of the predictions of a model due to them significantly deviating from the average of a data so in order to combat these processes such as the Z-score method would be utilised in their detection.

During the transformation of data, some features may be deemed to play a more significant role than others in the exploratory analysis of data. The higher quality of data outputted by this process the more ideal the results and predictions from a model can be. Once all these processes have been completed the suitable data splitting can be done, splitting the data into a percentage either for training the model or testing the model.

From the visual element, the output of this stage would provide marketing departments or data scientists with the visual data need to provide them valuable insight. On the other hand concerning model development, ML engineers would able to make use of the processed clean data for the development of their models.

Model Development

The next stage concerns the development of models. By using the clean data from the previous stage the next task would be to chose an suitable machine learning model to tackle a given problem. The development of a model can be broken down into a few elements (besides the selection of a model) consisting of training with either default or modified parameters for initial training of the model. This first fitting of the data may not result in an ideal outcome when the model is searching for patterns.

Once the model has been trained a simple evaluation can be performed on seen data to inspect the performance of the model taking into account metrics such as accuracy, precision and recall. These factors are not limited to the seen data but to the unseen data as they would provide insight into how well the model performs on real world data.

In some cases a validation set is used to fine tune a model's performance to further improve its results. This entire stage is an iterative process that has the data scientist repeating a process of improvement. The end result of this stage to provide the next stage being deployment is a functional and finalized model that can be used in a production environment.

Model Deployment

The goal of the model deployment stage is to integrate a model into an remote production environment that would allow a variety of authorised and invested entities to make decisions business or other wise based on the out of the model.

There are tools that already exist to optimize this process such as web APIs like FastAPI that is able to deploy models into an interface accessed via web. Requirements such as the portability and scalability of the model would be better supported by such a stage. Being able to ensure that a model would not have to be constantly redesigned is what scalability able to tackle while model portability is concerned with the access of it between a multitude of systems. The model deployment stage is worked on by software engineers in addition to data scientists.

Model Monitoring

This last stage places emphasis on the monitoring of a model. What this entails is the tracking and overviewing of the model's in production/real-time performance. Besides the standard performance metrics utilised during the development of the model the addition of confusion matrices and classification report would be able to provide far more valuable insight into the performance of said model.

The ability to compare all these metrics to previous ones in older developing versions of the model allow a timeline of progress be created would ensure that the model is in a constant state of iterative optimization of performance over time.

Insight in to if a model is experiencing any degradation or drift would allow for the immediate detection and swift prevention of such an occurrence. The overall goal of this stage is to identify when a model need to be finetune in order to maintain or improve its performance.

Implementation of Pipeline

Setting up of Environment

This section of the report will discuss the tools and methods used in setting up of the environment used to create the pipeline. Each tool, package or technique discussed here all serves a purpose. The pipeline will be built in Ubuntu 22.

Conda: was the first package to be installed as it is the corner stone of isolating and manging the environment that is going to be used for the development of the pipeline. It will help helps manage the appropriate versions of dependencies and packages. Python 3.8 is planned to be used within pipeline.

Docker Containers: this is the second element setup concerning the containerization of software, but it differs from conda in the fact that it essentially isolates applications ensuring that they are ran the same way in every setup environment. With this in mind it allows for simpler deployment in addition to the scalability of the pipeline. Redis was also installed in tandem as they will be used together to create in-memory data stores for the processes performed on the data.

Airflow: this is the platform that will be responsible for authoring, scheduling, and monitoring workflows. It will be automating and managing a variety of processes or dependencies that are in the pipeline. It will be the managing of the pipeline far simpler.

MLFlow: this is concerned with the life cycle of a model. it is an open-source platform that will manage any experimenting, reproduction or deployment of a model. its role in tracking and managing the varying versions of models.

Great Expectations: will act as the tool that aids in the defining and testing of data from a qualitative perspective. It provides the ability of defining expectations that the data should appear a particular way as well as automatically check if the data is able to hit set expectations. It will be utilised during the validation processes ensuring that data is kept to a suitable standard.

FastAPI: this was the last element to be installed on the system, as previously it before it is the web API that will be used in the deployment process of a model by providing a web-based interface for users to interact with.

Data ingestion

In this stage of developing the pipeline the first task is to ensure that the data in an accessible location. As the “stroke dataset ” was downloaded as a zip file from Kaggle and was placed in a folder called “data” which is located inside the documents folder or follow the absolute path of: ‘/home/user1/Documents/data/archive.zip’. The method of ETL was deemed the most appropriate as I am working with a smaller dataset that could possibly require more complex transformations during the pre-processing stage.

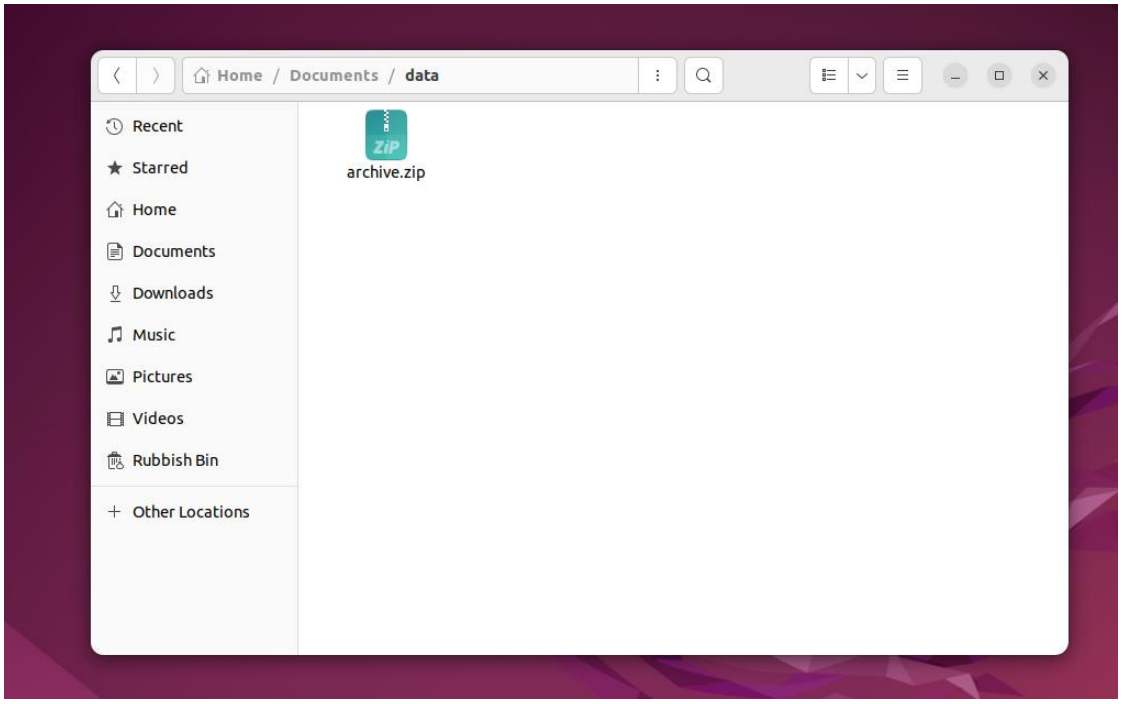


Figure 5: Stroke Dataset Zip File Downloaded

Now that the data has been acquired the next step is to set up the pipeline logistics that are going to be used by airflow such as pythons files for the functions that will be used in each part of the pipeline in addition to the DAG files that will connect everything together.

```
(pipeline_envo) user1@user1-VirtualBox:~/airflow/pipeDags$ touch pipeline_func.py
```

Figure 6: Creation of 'pipefunc.py'

```
(pipeline_envo) user1@user1-VirtualBox:~/airflow/pipeDags$ touch pDag.py
```

Figure 7: Creation of 'pDag.py'

The two files created that were placed in the folder structure ”/home/airflow/pipeDags/”:

- ‘pipeline_func.py’ = Holds all functions used by DAG
- ‘pDag.py’ = Holds all DAG code

All following code is from the ‘pipe_func.py’:

```
user1@user1-VirtualBox: ~/airflow/pipeDags
 GNU nano 6.2                                    pipe_func.py
# Responsible for pipeline logic
import sys
import pandas as pd
import redis
import pyarrow as pa
import numpy as np
import zipfile
import pickle as pk
import great_expectations as ge
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from urllib.parse import urlparse
from sklearn.svm import SVC
import mlflow
import mlflow.sklearn
import logging
from imblearn.over_sampling import SMOTE
```

Figure 8: All necessary import (pipe_func.py)

```python
# Unzipping Data

def extc_csv():

        try:
                with zipfile.ZipFile("/home/user1/Documents/data/archive.zip", 'r') as zip_ref:
                        zip_ref.extractall('/home/user1/Documents/data/')
        except Exception as e:
                logger.exception('''Error: Unable to unzip files. %s''', e)

# Data to DataFrame
        try:
                df = pd.read_csv('/home/user1/Documents/data/healthcare-dataset-stroke-data.csv')
                return (df)
        except Exception as e:
                logger.exception('''Error: unable to access datasets.''', e)

# Data Ingestion
def ingest(redis_conn, serialisation_context):
        def inner():
                df_s = extc_csv()
                df_serial = df_s

                df_serial.to_csv("~/Documents/data/stroke_data.csv")
                df_serial = pd.read_csv('~/Documents/data/stroke_data.csv')

                serialized_data =serialisation_context.serialize(df_serial).to_buffer().to_pybytes()
                redis_conn.set("stroke_ingested", serialized_data)
        return inner
```

Figure 9: Code for base ingestion (pipe_func.py)

This code performs the process of extracting the CSV from the downloaded zip file with the addition of a rename to ‘stroke_data.csv’ and it is then serialised using redis. The next code below is from the DAG used for shaping the pipeline from ‘pDag.py’:

```
user1@user1-VirtualBox: ~/airflow/pipeDags
  GNU nano 6.2                                                pDag.py
from datetime import timedelta, datetime
from airflow import DAG

from airflow.operators.python_operator import PythonOperator
from airflow.operators.bash import BashOperator

import airflow
import mlflow as ml
import mlflow.sklearn
import logging

import pandas as pd
import numpy as np
import redis
import pyarrow as pa
from pipe_func import ingest
```

Figure 10: Importing for DAG (pDag.py)

```
with DAG(
        "pDag",
        default_args=default_args,
        description="Pipeline DAG",
        schedule_interval=timedelta(days=1),
        start_date=airflow.utils.dates.days_ago(1),
        catchup=False,
        tags=["Pipeline"],
) as dag:
        ingestion_task = PythonOperator(
                task_id="ingestion",
                python_callable=ingest(red_con, serial_context)
        )
```

Figure 11:DAG Task Setup 'ingestion'

Validating Dataset

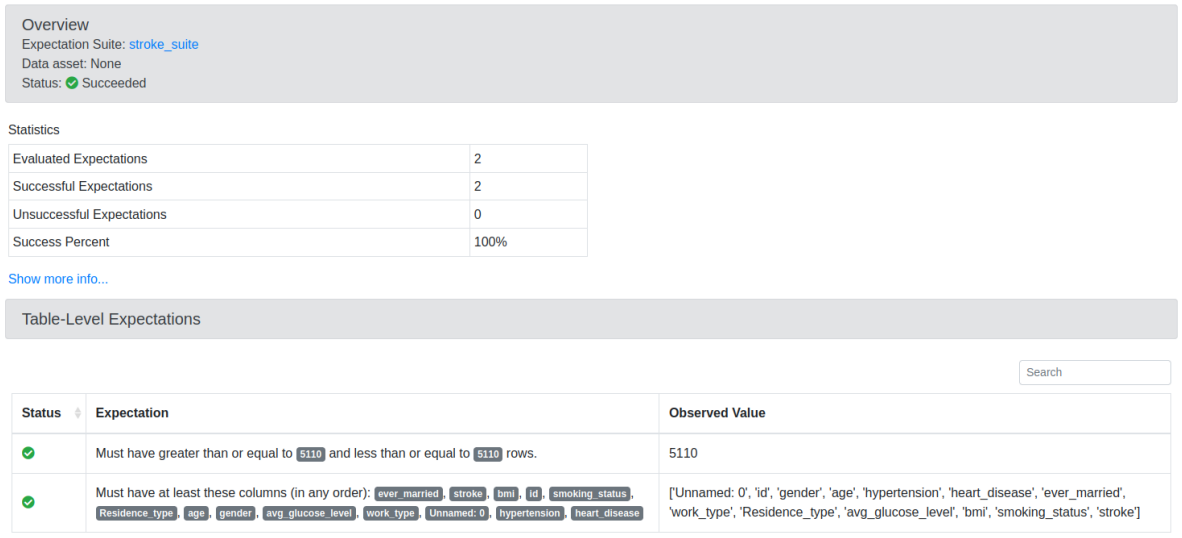
Now that all the this code is set up, extra checks for validating the data can now be performed. The structure and values of the dataset can be validated. A general check would be performed through the use of jupyter notebook to inspect the data set:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         5110 non-null   int64  
 1   id                 5110 non-null   int64  
 2   gender             5110 non-null   object 
 3   age                5110 non-null   float64
 4   hypertension       5110 non-null   int64  
 5   heart_disease      5110 non-null   int64  
 6   ever_married       5110 non-null   object 
 7   work_type          5110 non-null   object 
 8   Residence_type     5110 non-null   object 
 9   avg_glucose_level  5110 non-null   float64
 10  bmi                4909 non-null   float64
 11  smoking_status     5110 non-null   object 
 12  stroke             5110 non-null   int64  
dtypes: float64(3), int64(5), object(5)
memory usage: 519.1+ KB
```

Figure 12: DataFrame Inspection

After running great expectations we can if any error may have been encountered.



Overview
Expectation Suite: stroke_suite
Data asset: None
Status: Succeeded

Statistics

Evaluated Expectations	2
Successful Expectations	2
Unsuccessful Expectations	0
Success Percent	100%

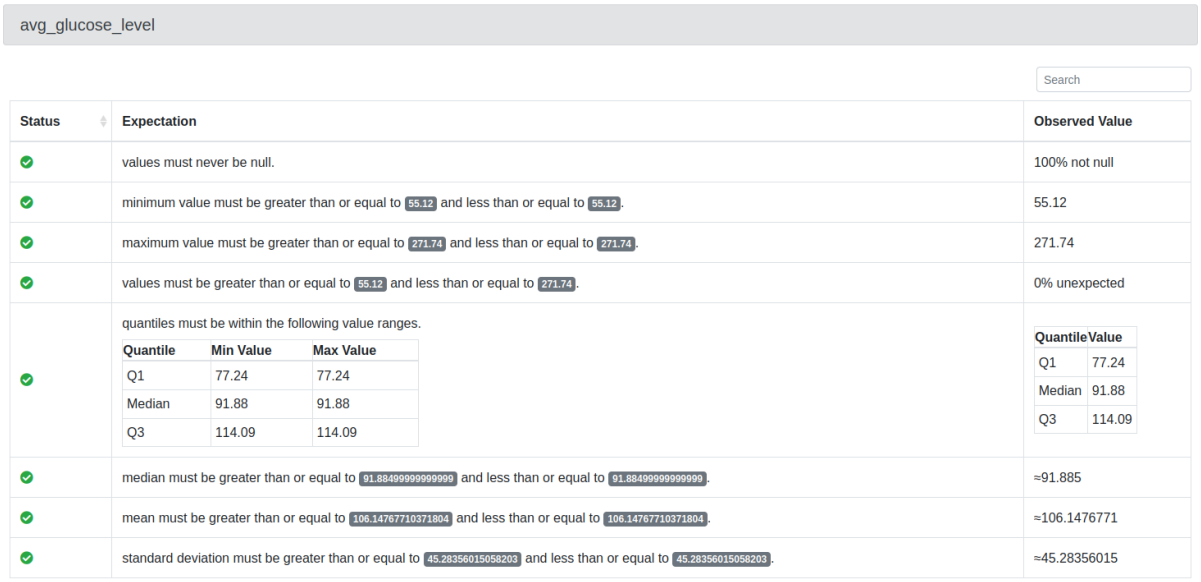
Show more info...

Table-Level Expectations

Search

Status	Expectation	Observed Value
✓	Must have greater than or equal to 5110 and less than or equal to 5110 rows.	5110
✓	Must have at least these columns (in any order): ever_married, stroke, bmi, id, smoking_status, Residence_type, age, gender, avg_glucose_level, work_type, Unnamed: 0, hypertension, heart_disease	['Unnamed: 0', 'id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'Residence_type', 'avg_glucose_level', 'bmi', 'smoking_status', 'stroke']

Figure 13: Great Expectations of Table



avg_glucose_level

Search

Status	Expectation	Observed Value
✓	values must never be null.	100% not null
✓	minimum value must be greater than or equal to 55.12 and less than or equal to 55.12.	55.12
✓	maximum value must be greater than or equal to 271.74 and less than or equal to 271.74.	271.74
✓	values must be greater than or equal to 55.12 and less than or equal to 271.74.	0% unexpected
✓	quantiles must be within the following value ranges. Quantile / Min Value / Max Value Q1 / 77.24 / 77.24 Median / 91.88 / 91.88 Q3 / 114.09 / 114.09	Quantile / Value Q1 / 77.24 Median / 91.88 Q3 / 114.09
✓	median must be greater than or equal to 91.88499999999999 and less than or equal to 91.88499999999999.	≈91.885
✓	mean must be greater than or equal to 106.14767710371804 and less than or equal to 106.14767710371804.	≈106.1476771
✓	standard deviation must be greater than or equal to 45.28356015058203 and less than or equal to 45.28356015058203.	≈45.28356015

Figure 14:Great Expectations of avg_glucose_level

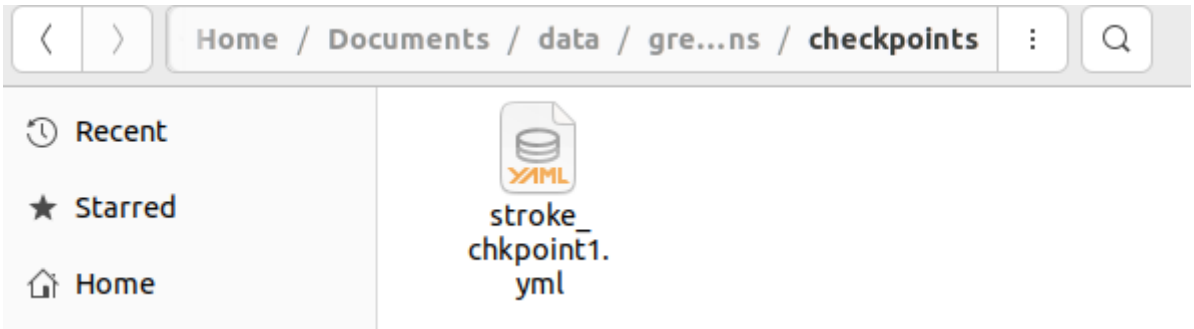


Figure 15: Checkpoint for later access to Expectations

Data Pre-processing

This section is concerned with the utilising the validated data to perform any necessary cleaning transformation or modifications to the data. The encoding of categorical values is the first task to be completed as there are a few categorical features present in the dataset.

Data encoding:

- Gender
- Work type
- Residence type
- Smoking status
- Ever married.

These features are the only categorical ones present in the dataset so the use of the 'get_dummies()' function was used to perform simple encodings. After theses encoding have been appropriately encoded the suitable feature selection can take place by removing irrelevant data.

The features that were chosen to be kept are display in figure 16. Features such as 'id','gender_other' and 'Unnamed' are examples of features that would not add any insight or value to the development of a model in the later parts of the pipeline.

```
# Data Pre-Processing
def preprocess(redis_conn, serialisation_context):
        def inner():
                df_fl = serialisation_context.deserialize(redis_conn.get("stroke_validate"))

        # encoding categorical features
                df_fl = pd.get_dummies(df_fl, columns=['gender','work_type','Residence_type','smoking_status'])
                df_fl = pd.get_dummies(df_fl, drop_first=True, columns=['ever_married'])

        # dropping appropriate features
                df_fl = df_fl.drop(labels=['id','gender_Other'], axis=1)

        # interpolating missing values
                df_fl = df_fl.interpolate(method='linear')

        # feature selection
                df_fl = df_fl[['age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status_Unknown','smoking_status_formerly smoked','smoking_status_never smoked',

                # splitting data
                tts = trn_tst_slt(df_fl)

        # serialising data to redis
                serial_dt = serialisation_context.serialize(tts).to_buffer().to_pybytes()
                redis_conn.set("stroke_prepared", serial_dt)
        return inner
```

Figure 16: Code for pre-processing

Interpolation was another task that was required to be performed during the pre-processing due to the 'bmi' feature presenting with 201 missing values in figure 17.

```
In [8]: df.isnull().sum()

Out[8]: Unnamed: 0          0
        id                  0
        gender              0
        age                 0
        hypertension        0
        heart_disease       0
        ever_married        0
        work_type           0
        Residence_type      0
        avg_glucose_level   0
        bmi               201
        smoking_status      0
        stroke              0
        dtype: int64
```

Figure 17: table of missing values

The final step of the pre-processing stage in the pipeline is the splitting of data. The data was split into training and testing in a percentage of 70&:30% and is reference in figure 18.

```
# Data Splitting
def trn_tst_slt(data):
        X = data.drop(labels=['stroke'], axis=1)
        y = data['stroke']

        X_trn, X_tst, y_trn, y_tst = train_test_split(X,y, train_size=0.7, test_size=0.3, random_state=20)

        return (X_trn, y_trn, X_tst, y_tst)
```

Figure 18: Splitting of data function.

The task id for the pre-processing is presented in DAG file in figure 19.

```
with DAG(
        "pDag",
        default_args=default_args,
        description="Pipeline DAG",
        schedule_interval=timedelta(days=1),
        start_date=airflow.utils.dates.days_ago(1),
        catchup=False,
        tags=["Pipeline"],
) as dag:
        ingestion_task = PythonOperator(
                task_id="ingestion",
                python_callable=ingest(red_con, serial_context)
        )

        validation_task = PythonOperator(
                task_id="validation",
                python_callable=valid(red_con, serial_context),
        )

        preprocess_task = PythonOperator(
                task_id="preprocessing",
                python_callable=preprocess(red_con, serial_context),
        )
```

Figure 19: Pre-processing added to DAG

Model Development

With the stroke data being completely pre-processed we are now able to input that data into a model which brings us to the model development stage. In the model training function it breaks the dataset into the training and testing variables respectively. In addition to this MLFlow and the SVM is initiated for its use in this stage. The model is then fitted with the training data and then serialized.

```
# Model Training
def mtrain(redis_conn, serialisation_context):
        def inner():
                allSets = serialisation_context.deserialize(redis_conn.get("stroke_prepared"))

                X_trn = allSets[0]
                y_trn = allSets[1]
                X_tst = allSets[2]
                y_tst = allSets[3]

        # mlflow initialisaton
                run = mlflow.start_run()

        # model initialisation
                clf = SVC(kernel='linear')

                # model fitting
                clf.fit(X_trn, y_trn)

        # model and data serialisation
                serial_clf = pk.dumps(clf)
                serial_run_id = serialisation_context.serialize(run.info.run_id).to_buffer().to_pybytes()

        # model stored in redis
                redis_conn.set("stroke_trained_clf", serial_clf)

        # model for furthur use
                redis_conn.set("stroke_trained_run_id", serial_run_id)
        return inner
```

Figure 20: Model training and fitting

```
# Model Metrics
def model_metrics(act, pred):

    # accuracy score
        acc = accuracy_score(act,pred)
    # precision
        precis = precision_score(act,pred, average='macro')
    # recall
        recall = recall_score(act,pred, average='macro')
    # f1 score
        f1 = f1_score(act,pred,average='macro')

        return acc, precis, recall, f1
```

Figure 21: Metric for Model Performance

```python
# Model Testing
def mtest(redis_conn, serialisation_context):
        def inner():
                # getting model
                clf = pk.loads(redis_conn.get("stroke_trained_clf"))

        # getting run id
                rn_id = serialisation_context.deserialize(redis_conn.get("stroke_trained_run_id"))

        # getting variables
                t_data = serialisation_context.deserialize(redis_conn.get("stroke_prepared"))
                X_tst = t_data[2]
                y_tst = t_data[3]

        # continuing previous experiment
                rn = mlflow.start_run(rn_id)

        # performing predictions
                pred = clf.predict(X_tst)

        # Metric Evaluation
                acc, precis, recall, f1 = model_metrics(y_tst, pred)

        # metric output
                sys.stdout.write("Accuracy : %s\n" % acc)
                sys.stdout.write("Precision: %s\n" % precis)
                sys.stdout.write("Recall   : %s\n" % recall)
                sys.stdout.write("F1 Score : %s\n" % f1)#

        # Logging in mlflow
                mlflow.log_metric("Accuracy", acc)
                mlflow.log_metric("Precision", precis)
                mlflow.log_metric("Recall", recall)
                mlflow.log_metric("F1", f1)
                tracking_url_type_filestore = urlparse(mlflow.get_tracking_uri()).scheme

                if tracking_url_type_filestore != "file":
                        mlflow.sklearn.log_model(clf, "model", registered_model_name="Stroke_Predication")
                else:
                        mlflow.sklearn.log_model(clf, "model")

                mlflow.end_run()

        return inner
```

Figure 22: Model Testing and MLFlow Logging

```python
with DAG(
        "pDag",
        default_args=default_args,
        description="Pipeline DAG",
        schedule_interval=timedelta(days=1),
        start_date=airflow.utils.dates.days_ago(1),
        catchup=False,
        tags=["Pipeline"],
) as dag:
        ingestion_task = PythonOperator(
                task_id="ingestion",
                python_callable=ingest(red_con, serial_context)
        )

        validation_task = PythonOperator(
                task_id="validation",
                python_callable=valid(red_con, serial_context),
        )

        preprocess_task = PythonOperator(
                task_id="preprocessing",
                python_callable=preprocess(red_con, serial_context),
        )

        training_task = PythonOperator(
                task_id="training",
                python_callable=mtrain(red_con, serial_context),
        )

        testing_task = PythonOperator(
                task_id="testing",
                python_callable=mtest(red_con, serial_context),
        )

        ingestion_task >> validation_task >> preprocess_task >> training_task >> testing_task
```

Figure 23: Implementing Training and Testing to DAG

Figure 24 displays how all the segments appear in the data pipeline when completely setup and functioning. With all theses stage complete it just a matter of deployment of the model which takes us to our next stage.

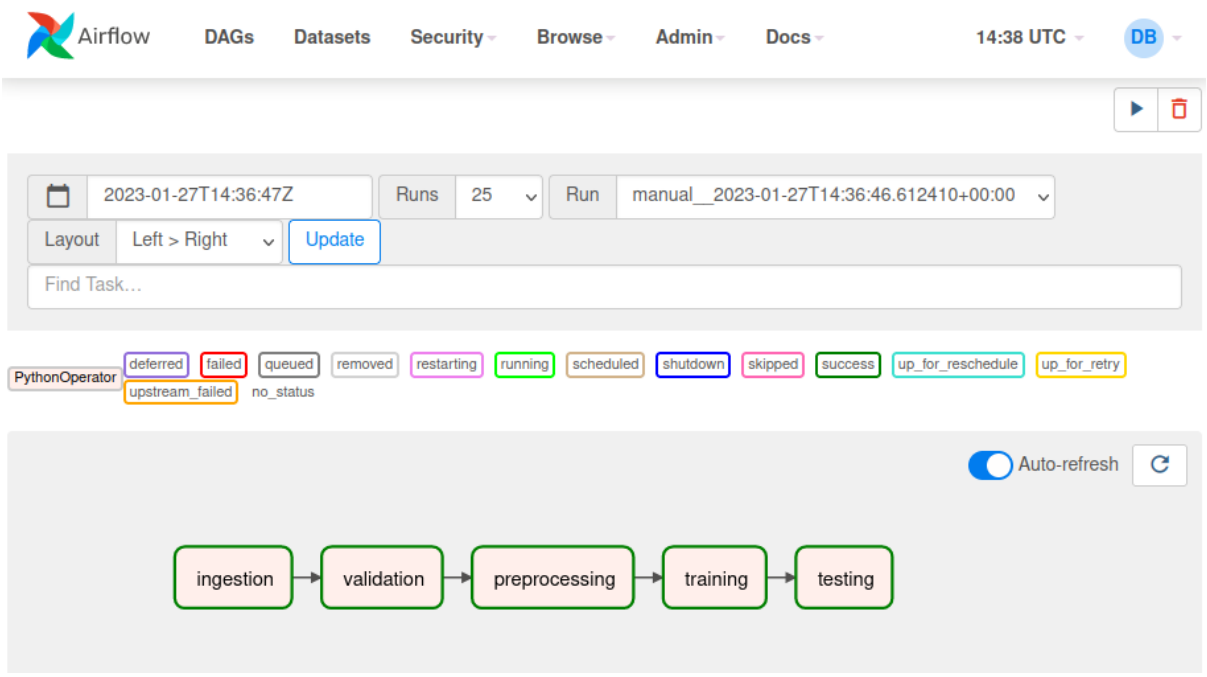


Figure 24: Airflow Data Pipeline

Model Deployment

As previously mentioned this last stage places emphasis on the monitoring of a model. What this entails is the tracking and overviewing of the model's in production/real-time performance. Besides the standard performance metrics utilised during the development of the model the addition of confusion matrices and classification report would be able to provide far more valuable insight into the performance of said model.

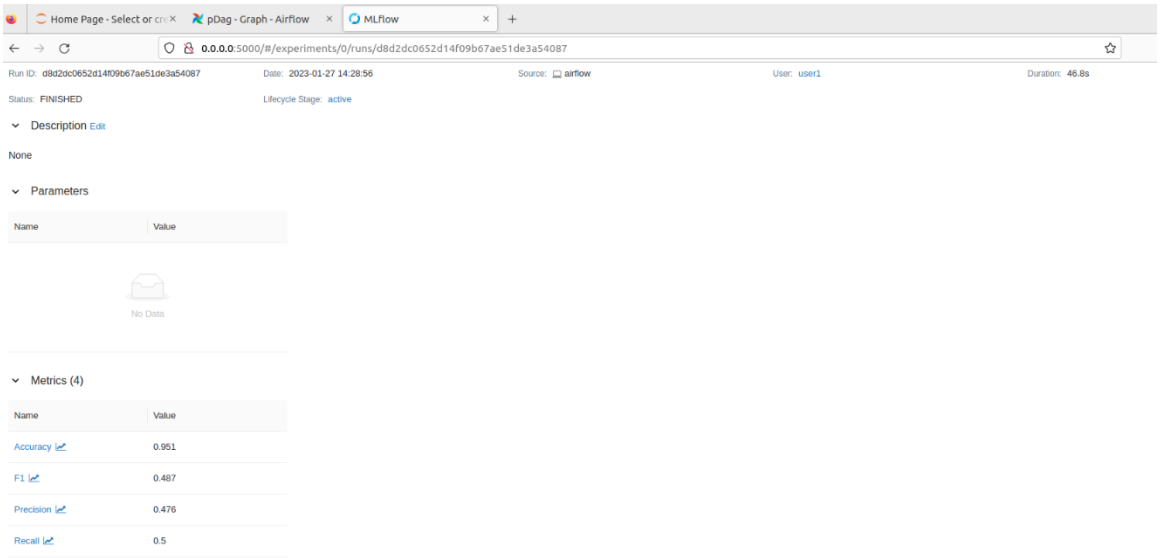


Figure 25: MLFlow Displaying metrics

The alternative to accessing elements of the model is with the use of FastAPI it is able to deploy models into an interface accessed via web. With the benefit of portability and scalability of the model being able to ensure that a model would not have to be constantly redesigned is what scalability able to tackle while model portability is concerned with the access of it between a multitude of systems.

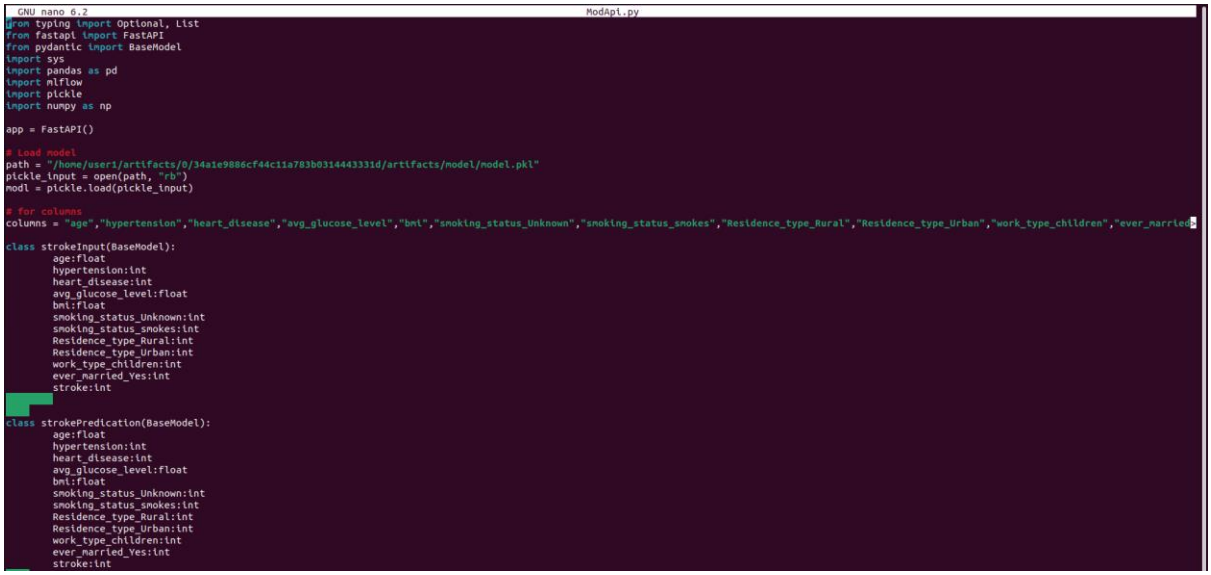


Figure 26: FastAPI Setup File

Figure 27: Defined classes use to output values

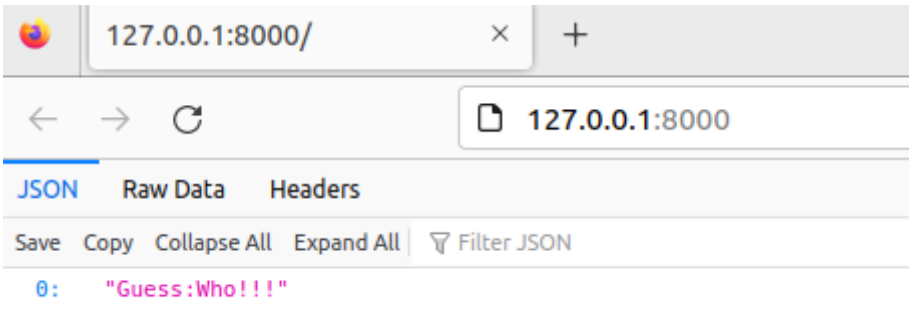


Figure 28: Main API Page

Model Monitoring

This last stage places emphasis on the monitoring of a model. What this entails is the tracking and overviewing of the model's in production/real-time performance. Besides the standard performance metrics utilised during the development of the model the addition of confusion matrices and classification report would be able to provide far more valuable insight into the performance of said model. With the endpoint created inputting http://127.0.0.1:8000/docs into our URL will display the following web page:

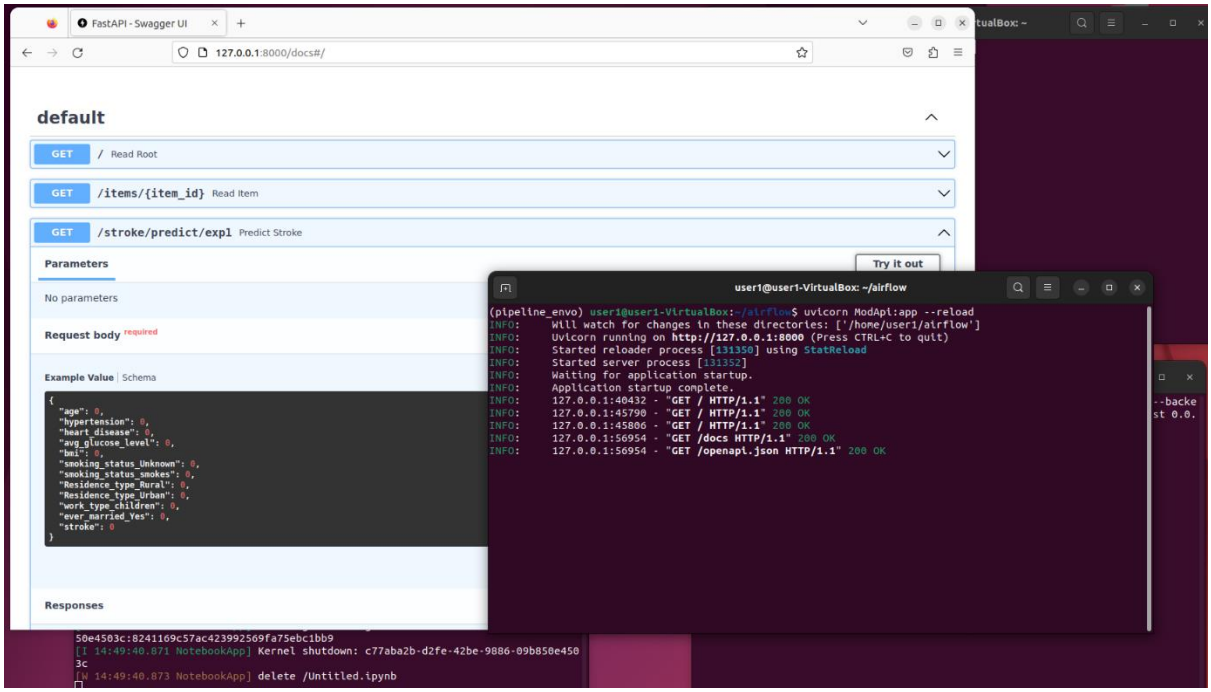


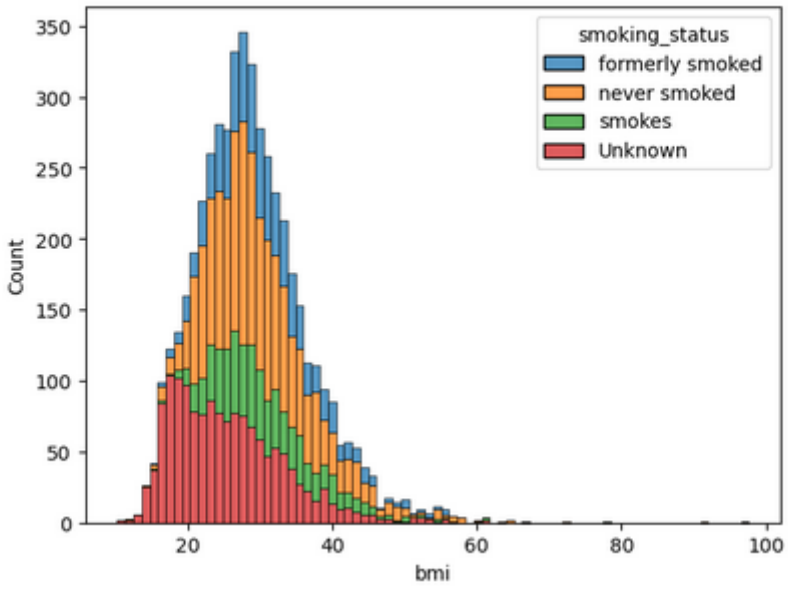
Figure 29:Predication being send via query

Insight from Analysis

In the exploratory data analysis the core library used for much of the visualisations was seaborn due to its similarity to matplotlib. A mixture of histplot, scatterplot and boxplots were used throughout the EDA. From the EDA performed it was established that 'genders' are quite evenly distributed between 'male' and 'female' but there is only 1 record for the 'other' gender type which may present problems for use in models later on. The 'ever married' feature presents that more people are married than not. From the 'work type' feature those that work in the private sector were the most prominent.

The 'smoking status' is quite evenly distributed, with 'ever smoked' having the greatest number of records. Most of the strokes are appeared to be happening to those between the age gap of 40 and 80, with the age of 80 having the highest number of results. Most of the patients have around 30 BMI score but, most of the strokes happened to the patients with a score of 20 and 40. Interestingly those that are married present seem to present with more strokes and when the married data is split through the feature of gender, males seem to have a higher occurrence of stroke than female when married. The reverse appears with females that are not married. This may occur for a multitude of reasons but is unfortunately beyond the scope of this report. Visualisation are presented below:

```
In [5]: sns.histplot(data=df, x="bmi", hue="smoking_status", multiple="stack")
Out[5]: <AxesSubplot:xlabel='bmi', ylabel='Count'>
```




```
In [6]: sns.histplot(data=df, x="smoking_status", hue="stroke", multiple="stack")
Out[6]: <AxesSubplot:xlabel='smoking_status', ylabel='Count'>
```

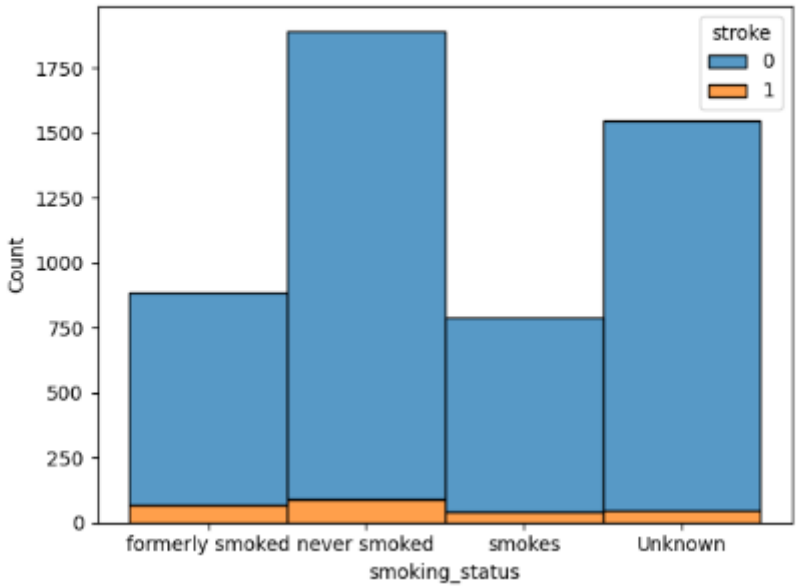
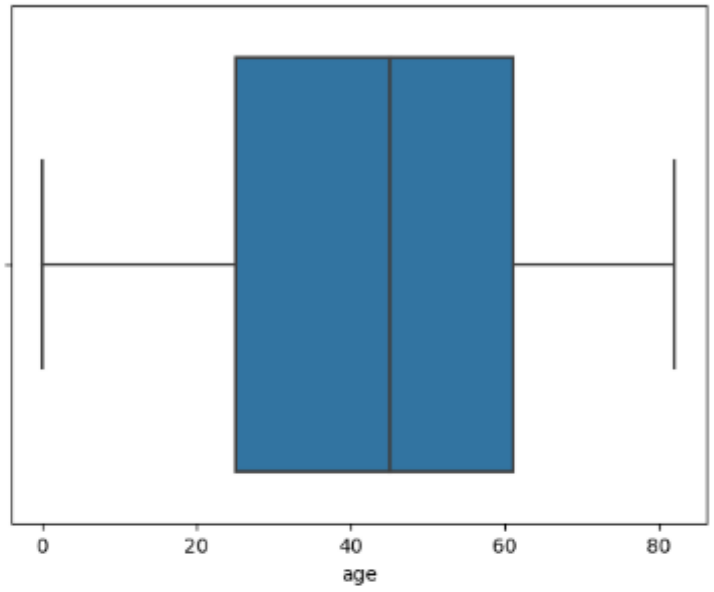


Figure 30: EDA Visualisation 2 of 6

```
In [9]: sns.boxplot(data=df, x="age")
Out[9]: <AxesSubplot:xlabel='age'>
```




```
In [10]: sns.histplot(data=df, x="ever_married")
Out[10]: <AxesSubplot:xlabel='ever_married', ylabel='Count'>
```

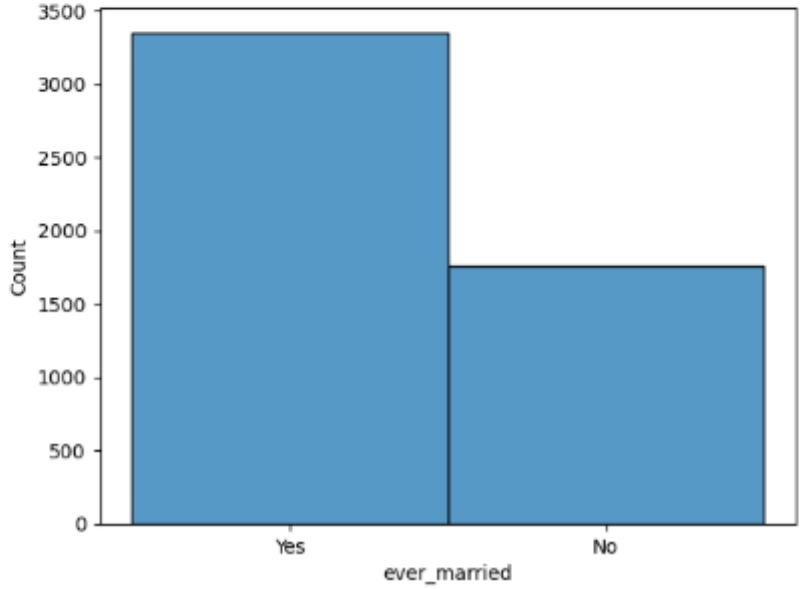
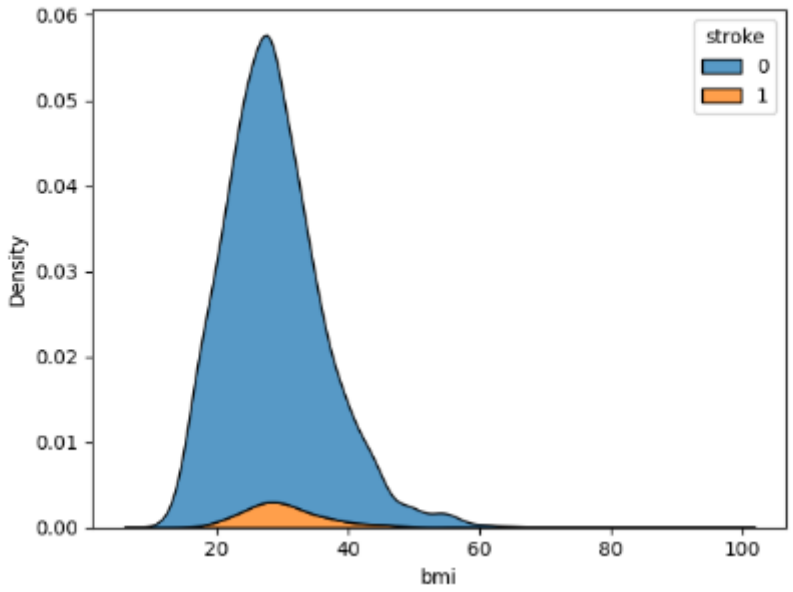


Figure 31: EDA Visualisation 4 of 6

```
In [7]: sns.kdeplot(data=df, x="bmi", hue="stroke", multiple="stack")
Out[7]: <AxesSubplot:xlabel='bmi', ylabel='Density'>
```




```
In [8]: sns.histplot(data=df, x="work_type")
Out[8]: <AxesSubplot:xlabel='work_type', ylabel='Count'>
```

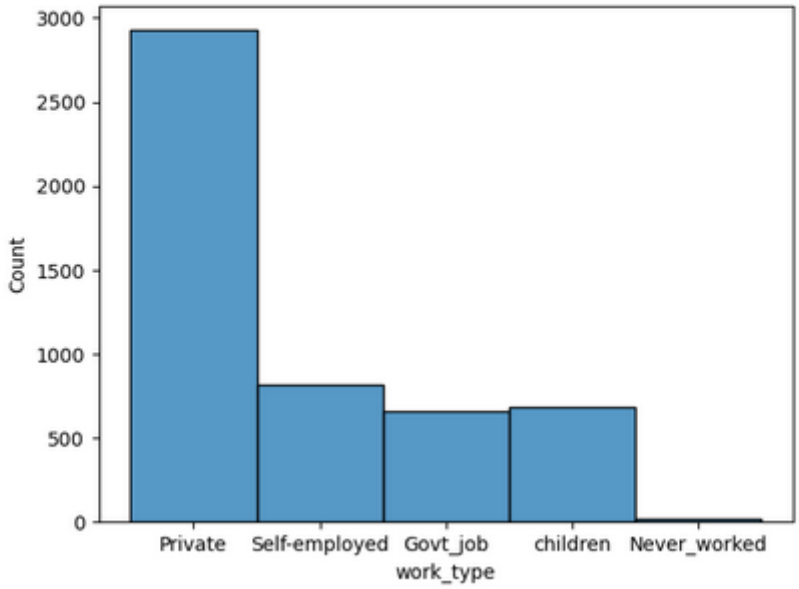


Figure 32: EDA Visualisation 6 of 6

Concerns of Data Security and Legal Dilemmas

Data Privacy

When Data privacy is brought up most people would rightfully assume that it means the capability of a person decide to what level of their personal data is shared in addition to who has access to it. Such personal data could be an individual's name geographical location or contact data.

Similar to how someone might want to exclude some people from a private discussion, many online users want to control or limit the collection of particular types of personal data. The importance of data privacy has increased along with the use of the Internet.

Websites, programmes, and social media platforms commonly need to collect and store personal data about users in order to provide services. However, some platforms and applications may collect and use user data in ways that go beyond what customers had anticipated, offering users less privacy than they had intended. The data that other platforms and apps collect could not have the appropriate security measures in place, which could result in a data breach that violates user privacy.

Data Security

Data security refers to the measures taken to keep data secure against unauthorised access and to preserve its confidentiality, integrity, and availability. Aside from data protection techniques like encryption, key management, data redaction, data subletting, and data masking, best practises for data security also include privileged user access limits, auditing, and monitoring.

Data Ethics

This theme issue's primary goal is to establish data ethics as a new field of ethics that investigates and assesses ethical issues related to data algorithms and related practises (such as responsible innovation, programming, hacking, and professional codes), with the goal of developing and promoting morally sound solutions. Data ethics expands upon the theoretical groundwork established by computer and information ethics while also improving the methodology supported up to this point in this field of study by changing the level of abstraction of ethical inquiries from being information-centric to being data-centric.

Data Protection Law

It is increasingly common for personal details to be stored on computers. The Data Protection Act exists to protect such details. This personal data includes items such as:

- name and address
- date of birth
- medical records
- school and employment records

Only authorised individuals should have access to personal information because it is private. Additionally, accessing, copying, and sharing digital files saved on computers can be simple. To ensure that our personal information is kept secure, unaltered, and not deleted, protection is required. To guarantee that personal data is appropriately protected, the Data Protection Act was created. Additionally, everyone has the right to request that any organisation that holds personal information on them erase or correct it if it is inaccurate.

GDPR

On May 25, 2018, the General Data Protection Regulation (GDPR), a rule of the European Union (EU), went into effect. It is intended to tighten and harmonise data protection for persons inside the EU and supersedes the 1995 EU Data Protection Directive. The GDPR lays out the guidelines for the collection, processing, storage, and use of personal data and grants individuals specific rights over their personal information, including the right to access, rectification, erasure, and restriction of processing. Regardless matter where an organisation is situated, it must comply with the rule if it handles the personal data of EU individuals.

Sources

kaggle.com. (n.d. b). Heart Disease Predication Data. [online]
Available at:
https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction?resource=download
[Accessed 20 Oct. 2022]

kaggle.com. (n.d. b). Stroke Predication Data. [online]
Available at:
https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset
[Accessed 20 Oct. 2022]

kaggle.com. (n.d. b). Diabetes Predication Data. [online]
Available at: https://www.kaggle.com/datasets/whenamancodes/predict-diabities
[Accessed 20 Oct. 2022]

MLflow. (n.d.). MLflow - A platform for the machine learning lifecycle. [online]
Available at: https://mlflow.org
[Accessed 20 Oct. 2022].

airflow.apache.org. (n.d.). Email Configuration — Airflow Documentation. [online]
Available at: https://airflow.apache.org/docs/apache-airflow/stable/howto/email-config.html
[Accessed 20 Oct. 2022].

Anaconda. (2018). Anaconda. [online]
Available at: https://www.anaconda.com
[Accessed 20 Oct. 2022]

Apache Airflow. (n.d.). Home. [online]
Available at: https://airflow.apache.org
[Accessed 20 Oct. 2022]

DigitalOcean. (n.d. a). How To Install the Anaconda Python Distribution on Ubuntu 20.04. [online]
Available at: https://www.digitalocean.com/community/tutorials/how-to-install-the-anaconda-python distribution-on-ubuntu-20-04
[Accessed 20 Oct. 2022]

docker-fastapi-projects.readthedocs.io. (n.d.). Uvicorn — Docker FastAPI projects 0.0.2

documentation. [online]
Available at: https://docker-fastapi-projects.readthedocs.io/en/latest/uvicorn.html
[Accessed 20 Oct. 2022]

docs.greatexpectations.io. (n.d.). Getting started with Great Expectations | Great Expectations. [online]
Available at: https://docs.greatexpectations.io/docs/tutorials/getting_started/tutorial_overview/
[Accessed 20 Oct. 2022]