

Pulse Vision

Team 11: Final Project Documentation

Bharagavi Venkata: 002724793

Bhakti Ukey: 002922939

Charu Singh: 001520297

Aakash Bhatt: 002791277

Table of Contents

1.	ABSTRACT	3
2.	INTRODUCTION	4
3.	METHODOLOGY	5
3.1.	Data Source	6
3.2.	Data Extraction	6
3.3.	Data Cleaning	7
3.4.	Data Preprocessing	8
3.5.	Data Validation	8
3.6.	Exploratory Data Analysis	9
3.7.	Data Transformation	13
3.8.	Model Training	17
3.9.	Model Interpretability	20
4.	Technical Specifications	21
4.1.	Airflow	21
4.2.	Streamlit	24
4.3.	FastAPI	26
4.4.	Pytest	28
4.5.	Docker	30
4.6.	Aws Services	30
4.7.	Git Actions	30
5.	Conclusion	31
6.	REFERENCES	32

Abstract

Heart disease is a significant global health challenge that claims millions of lives every year and exerts a significant financial burden on the economy. In the United States, it is among the most prevalent chronic diseases, impacting millions of Americans annually. It is a severe medical condition that requires immediate attention. However, many of the risk factors associated with heart disease are preventable through simple lifestyle changes, such as adopting heart-healthy eating habits, engaging in regular exercise, and monitoring key health indicators like blood pressure, cholesterol, and weight. Understanding the relationship between heart disease and individual lifestyle choices and biological characteristics has far-reaching implications for healthcare and social policies, public awareness, and the promotion of health and wellness in communities.

The goal of this project is to analyze the correlation between heart disease and various lifestyle factors, with the aim of identifying actionable insights that can help individuals and healthcare providers reduce the incidence of heart disease and improve overall health outcomes. Pulse-Vision is a must-have tool for anyone looking to make meaningful changes to their lifestyle and reduce their risk of heart disease. Its innovative approach to health and wellness has the potential to transform the healthcare industry and improve the lives of millions of people worldwide.

Introduction

To address this pressing issue, our team has developed Pulse-Vision, a web application that leverages Machine Learning techniques to identify risk factors for heart diseases. Pulse-Vision provides invaluable insights into how prone individuals are to heart disorders by analyzing the correlation between lifestyle choices and known heart disorders. The app's user-friendly interface guides users through a comprehensive questionnaire that covers important factors that can be the risk factors leading to higher chances of getting heart illnesses. Using this information, Pulse-Vision calculates a personalized risk factor percentage, enabling users to take proactive measures to improve their heart health and live healthier, happier lives.

3. Methodology

In this section, we will describe the methodologies used in this project in detail. The purpose of this is to provide readers with a clear understanding of the process we followed and to facilitate replication of our work if desired.

3.1. Data source

The Behavioral Risk Factor Surveillance System (BRFSS) is a collaborative project between all the states in the United States and participating US territories and the Centers for Disease Control and Prevention (CDC). The BRFSS is administered and supported by CDC's Population Health Surveillance Branch, under the Division of Population Health at the CDC's National Center for Chronic Disease Prevention and Health Promotion. The BRFSS is a system of ongoing health-related telephone surveys designed to collect data on health-related risk behaviors, chronic health conditions, health-care access, and use of preventive services from the noninstitutionalized adult population (≥ 18 years) residing in the United States and participating areas. The BRFSS was initiated in 1984, with 15 states collecting surveillance data on risk behaviors through monthly telephone interviews. Over time, the number of states participating in the survey increased; BRFSS now collects data in all 50 states as well as the District of Columbia and participating US territories. During 2021, all 50 states, the District of Columbia, Guam, Puerto Rico, and US Virgin Islands collected BRFSS data. In this document, the term "state" refers to all areas participating in the BRFSS, including the District of Columbia, Guam, the Commonwealth of Puerto Rico, and the US Virgin Islands. Florida was unable to collect BRFSS data over enough months to meet the minimum requirements for inclusion in the 2021 annual aggregate data set. BRFSS's objective is to collect consistent state-specific data on health risk behaviors, chronic diseases and conditions, access to health care, and use of preventive health services related to the leading causes of death and disability in the United States. Factors assessed by the BRFSS in 2021 included health status and healthy days, exercise, hypertension awareness, cholesterol awareness, chronic health conditions, arthritis, tobacco use, fruit and vegetable consumption, and healthcare access (core section). Optional Module topics for 2021 included prediabetes and diabetes, cognitive decline, caregiver, cancer survivorship (type, treatment, pain management), and sexual orientation/gender identity (SOGI).

The data can be found on CDC's website:

https://www.cdc.gov/brfss/annual_data/annual_2021.html

3.2. Data Extraction

The Dataextraction.py script, located in the Scripts folder of the Git repository, is designed to extract data from a zip file and convert it from .XPT format to .csv format for analysis. The data source is a zip file located at 'http://www.cdc.gov/brfss/annual_data/2021/files/LLCP2021XPT.zip', which contains data in .XPT format. The script reads the zip file and extracts the .XPT file within it. Then, using a conversion process, the data is transformed into .csv format, which is more suitable for further analysis. This script is a valuable tool for anyone interested in analyzing the data contained in the LLCP2021XPT.zip file.

3.3. Data Cleaning

The extracted data is then pulled into DataCleaning.ipynb file to perform data cleaning. The original dataset contains 438693 rows and 303 columns. ***Research in the field has identified the following as important risk factors for heart disease:*** blood pressure (high), cholesterol (high), smoking, diabetes, obesity, age, sex, race, diet, exercise, alcohol consumption, BMI, Household Income, Education, Health care coverage, and Mental Health. The selected features that matched the risk factors were identified through research of heart disease and through. To help understand what the variables meant, the response options, and original questions, I consult the BRFSS Codebook (2019,2020,2010).

First choose the dependent or response variable to be a calculated variable _MICHHD. This variable combined responses from individual participants to form a new calculated question: Respondents that have ever reported having coronary heart disease (CHD) or myocardial infarction (MI). In this way, we determine that any participant that has ever been diagnosed with heart disease or has had a heart attack will be considered as having heart disease. From there, we selected categories of risk factors and selected questions in the BRFSS that matched those risk factors. We chose features related to the following risk factor categories High Blood Pressure, High Cholesterol, Body Mass Index (BMI), Smoking, Other Chronic Health Conditions, Physical Activity, Diet, Alcohol Consumption, Health Care Access, General Health & Wellbeing, and Demographics.

Column Description and Questions asked in the survey:

- **HeartDisease:** Respondents that have ever reported having coronary heart disease (CHD) or myocardial infarction (MI).
- **BMI:** Body Mass Index (BMI).
- **Smoking:** Have you smoked at least 100 cigarettes in your entire life?
- **AlcoholDrinking:** Heavy drinkers (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week)

- **Stroke:** (Ever told) (you had) a stroke?
- **PhysicalHealth:** Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good? (0-30 days).
- **MentalHealth:** Thinking about your mental health, for how many days during the past 30 days was your mental health not good? (0-30 days).
- **DiffWalking:** Do you have serious difficulty walking or climbing stairs?
- **Sex:** Are you male or female?
- **AgeCategory:** 13-categories of age. ['18-24', '25-29', '30-34', '35-39', '40-44', '45-49', '50-54', '55-59', '60-64', '65-69', '70-74', '75-79', '80 or older']
- **Race:** Imputed race/ethnicity value.
- **Diabetic:** (Ever told) (you had) diabetes?
- **PhysicalActivity:** Adults who reported doing physical activity or exercise during the past 30 days other than their regular job.
- **GenHealth:** Would you say that in general your health is...
- **SleepTime:** On average, how many hours of sleep do you get in a 24-hour period?
- **Asthma:** (Ever told) (you had) asthma?
- **KidneyDisease:** Not including kidney stones, bladder infection or incontinence, were you ever told you had kidney disease?
- **SkinCancer:** (Ever told) (you had) skin cancer?

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 18 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   HeartDisease                          319795 non-null object
1   BMI_value                             319795 non-null float64
2   Smoking                               319795 non-null object
3   AlcoholDrinking                       319795 non-null object
4   Stroke                                319795 non-null object
5   PhysicalHealth                         319795 non-null float64
6   MentalHealth                          319795 non-null float64
7   DiffWalking                           319795 non-null object
8   Sex                                    319795 non-null object
9   AgeCategory                           319795 non-null object
10  Race                                   319795 non-null object
11  Diabetic                               319795 non-null object
12  PhysicalActivity                       319795 non-null object
13  GenHealth                             319795 non-null object
14  SleepTime                             319795 non-null float64
15  Asthma                                319795 non-null object
16  KidneyDisease                          319795 non-null object
17  SkinCancer                            319795 non-null object
dtypes: float64(4), object(14)
memory usage: 43.9+ MB
```

For more information on these variables, I have included the original variable names and questions that were asked in the BRFSS in the Appendix Table 1. The variables mentioned above covered a wide range of risk factors and contain a higher number of data points than have been explored in past publications involving machine learning techniques applied to heart disease prediction.

3.4. Data Preprocessing

Once the appropriate features based on risk factor category were selected, a dataset of 438,693 responses with the dependent variable and 26 features. As part of the data cleaning process, all the missing values were removed, bringing the total number of responses down to 85438. Then individually all the variables were modified to be binary 0=No, and 1=Yes. Additionally, responses that were marked 'Don't Know/Not Sure' and 'Refused to Answer' were removed. Ordinal variables were changed to a scale '0,1,2,3,4....' for simplicity. To see all these modifications, visit the git repository (DataCleaning.ipynb). Finally, renamed all the features so it would be easier to interpret the results of feature selection during the model building process.

Following these pre-processing steps, a dataset with 39452 responses to 26 features and 1 response variable for HeartDisease.

The dataset initially consisted of 438,693 responses with 26 features and a dependent variable based on risk factor category. To ensure data quality, missing values were removed, resulting in 85438 responses. Each variable was transformed into binary format (0=No, and 1=Yes), and responses marked as 'Don't Know/Not Sure' and 'Refused to Answer' were removed. Ordinal variables were scaled for ease of interpretation (to a scale '0,1,2,3,4....'). Furthermore, feature names were renamed to improve clarity during feature selection. Ultimately, a dataset of 39452 responses with 26 features and one response variable, HeartDisease, was obtained.

3.5. Data Validation

Great Expectations

Great Expectations is a Python library that provides an easy way to create and automate data validation for data pipelines. It automatically profiles data and generates expectations based on the observed statistics. Great Expectations tells you whether each Expectation in an Expectation Suite passes or fails, and returns any unexpected values that failed a test, which can significantly speed up debugging data issues.

The cleaned data is used for automating the data validation process.

Steps for Data Validation:

1. Environment Setup and activation
2. Create a folder named `great-expectation`
3. Install module `great_expectations`
4. Initialization at the base directory
5. Change working directory to the newly created directory, great_expectation
6. Import data into the current repo
7. Connect to the Datasource(data)

8. Launch Cli Datasource process
9. Input in the prompt: The first Jupyter notebook should opens

10. Launch CLI suite
11. Validate Data
12. Create Checkpoint

To view commands (great-expectation/README.md): <https://github.com/BigDataIA-Spring2023-Team-11/PulseVision.git>

To view the data docs for Great Expectations: https://final-data-validation.s3.amazonaws.com/uncommitted/data_docs/local_site/index.html

3.6. Exploratory Data Analysis

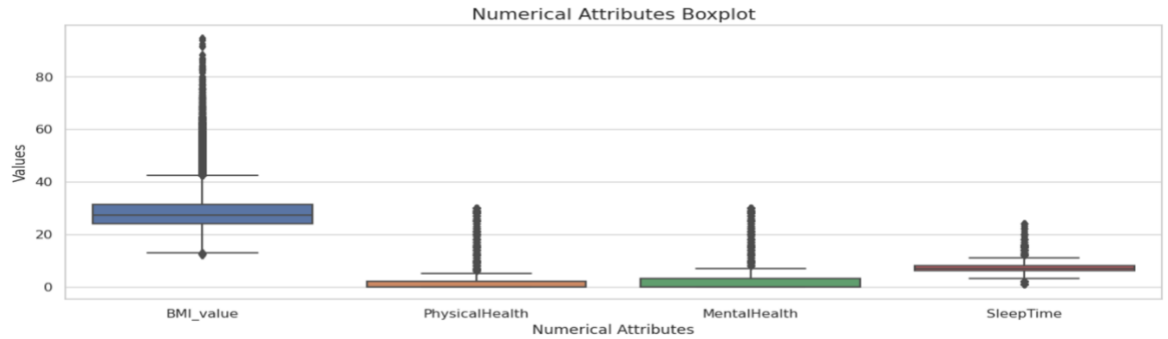
Exploratory Data Analysis (EDA) is a crucial step in the data analysis process. It involves performing initial investigations on the data to discover patterns, spot anomalies, and test hypotheses with the goal of understanding the data's underlying structure and characteristics.

3.6.1. Univariate Analysis:

1. Continuous Feature Analysis:

The Continuous Features in the dataset are columns named:

- BMI - Average BMI (Body Mass Index) of this dataset is 28.33 and most of the data is distributed between 12.02 to 31.42 with a maximum value of 94.85. There seems to be potential outlier in the dataset which we can investigate later in the notebook.
- PhysicalHealth and MentalHealth are following same distribution with small proportion of data in 1st and 2nd quantiles. Most of the people are not well for hardly 2 to 3 days but a portion of people are suffering from so many days.
- Sleep Time [Average Sleep Time] - As expected, the Average sleep time is around 7 hours with minimum of 1 hour and maximum of 24 hours which is shocking, definitely this is a data issue as no one can sleep for 1 hour or 24 hours on average per day.



2. Categorical Feature Analysis:

- The Categorical Features in the datasets are columns named: HeartDisease, Smoking, AlcoholDrinking, Stroke, DiffWalking, Sex, AgeCategory, Race, Diabetic, PhysicalActivity, GenHealth, Asthma, KidneyDisease, SkinCancer.
- Distribution of Smoking demonstrates that Smokers are more likely to get Heart Disease. The actual target distribution is around ~8.5% whereas the distribution of smokers with HeartDisease is around ~12%, so this feature might play a good role to identify Heart Disease, which we can confirm by calculating feature importance later in this notebook.
- Gender Distribution - shows there are More Male adults that have Heart Disease than Female peers.
- There are no noticeable differences between adults with and without heart disease in being a heavy drinker or having asthma.
- Nevertheless, people with heart disease seem to experience stroke and difficulty while walking more than those who don't.
- People who diagnosed with and without heart disease are comparably distinct in physical activity distribution, which seems like realistic as who are doing physical activity regularly are less prone to HeartDisease.
- There might be a strong correlation between increasing age and the presence of heart disease.
- Asian and Hispanic responders seem to have lower heart disease than other peers, but further analysis is necessary to confirm this statement.
- Adults with diabetes seem to have high rate of having heart diseases.

- Adults who considered Fair or Poor general health have higher chance of diagnosed with heart diseases.
- Distribution shows that adults with kidney disease or Skin Cancer are Highly Prone to Heart disease.

```

Features which has only 2 distinct values:
['HeartDisease', 'Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking', 'Sex', 'PhysicalActivity', 'Asthma', 'KidneyDisease', 'SkinCancer']

Features which has more than 2 distinct values:
['AgeCategory', 'Race', 'Diabetic', 'GenHealth']

AgeCategory --> ['55-59' '80 or older' '65-69' '75-79' '40-44' '70-74' '60-64' '50-54'
'45-49' '18-24' '35-39' '30-34' '25-29']

Race --> ['White' 'Black' 'Asian' 'American Indian/Alaskan Native' 'Other'
'Hispanic']

Diabetic --> ['Yes' 'No' 'No, borderline diabetes' 'Yes (during pregnancy)']

GenHealth --> ['Very good' 'Fair' 'Good' 'Poor' 'Excellent']

```

3.6.2. Outlier Handling:

1. Standard Deviation Method:

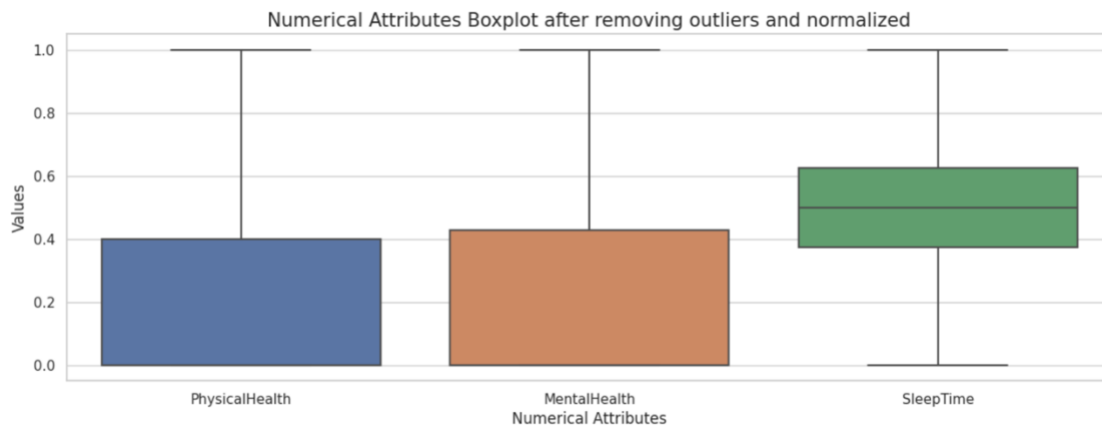
- Standard deviation is a metric of variance i.e. how much the individual data points are spread out from the mean. If a dataset approximately follows normal distribution, then around 68% of the data lies in 1 standard deviation from the mean, similarly ~95% of the data lies in 2 standard deviations from the mean and ~99.7% of the data lies in 3 standard deviations from the mean.

2. Inter Quantile Range Method:

- IQR is a concept in statistics that is used to measure the statistical dispersion and data variability by dividing the dataset into quartiles.
- $Q1$ (1st Quantile) = `df.quantile(0.25)`
- $Q3$ (3rd Quantile) = `df.quantile(0.75)`
- $IQR = Q3 - Q1$
- $LowerLimit = Q1 - 1.5 * IQR$
- $UpperLimit = Q3 + 1.5 * IQR$

we use 1.5 as a factor --> as Standard Deviation method, 1 IQR from Q1 & Q2 covers around ~70% of the data and 2 IQR covers around ~ 97% of the data, so bigger scale will lead to consider outliers as a data point and a smaller scale will lead to perceive some of the datapoints has outliers. so, it is approximated that 1.5 factor will cover around ~95% of the data.

Anything outside the LowerLimit and UpperLimit can be replaced with Median or Mean or Mode.



3.6.3. Missing Values:

- The data has been cleaned for missing and null values. Hence, There are no missing values in the dataset, so Imputation methods are not required.

```
Out [5]: HeartDisease      0
        BMI_value         0
        Smoking           0
        AlcoholDrinking   0
        Stroke            0
        PhysicalHealth     0
        MentalHealth      0
        DiffWalking       0
        Sex               0
        AgeCategory       0
        Race              0
        Diabetic           0
        PhysicalActivity   0
        GenHealth         0
        SleepTime         0
        Asthma            0
        KidneyDisease     0
        SkinCancer        0
        dtype: int64
```

3.7. Data Transformation

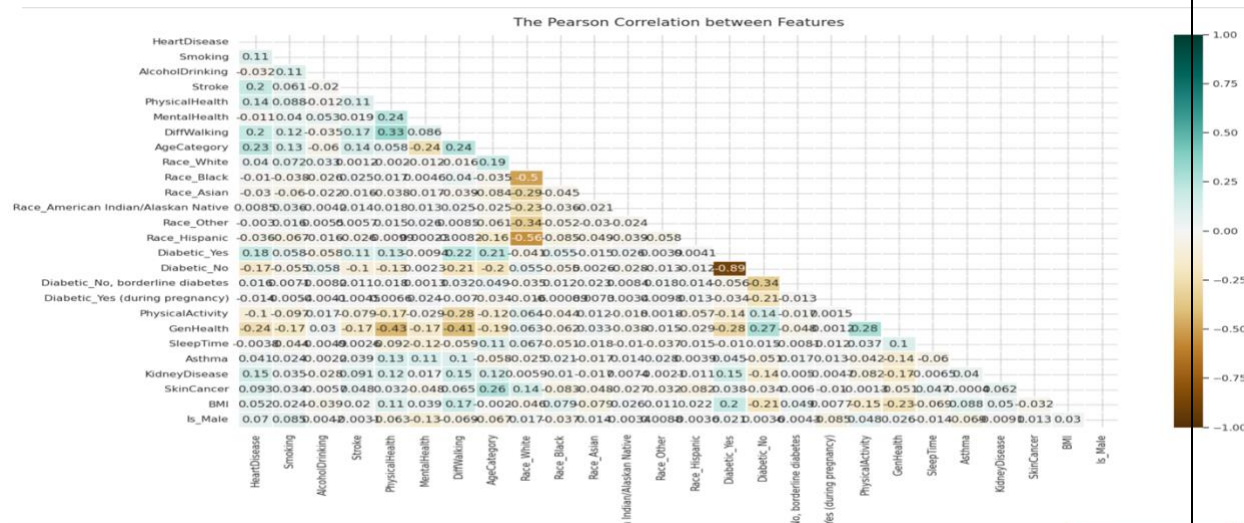
3.7.1. Encode Categorical Data using Dummies, and Encoders

- **Ordinal Encoder** for Age, Health, BMI: These categories have ordered relationship between each category, so Ordinal Encoder does that job by using given ordered map.
- **OneHotEncoder** for Race and Diabetic : These feature values do not have any order among each other and few categories have high impact on identifying heart Disease like Diabetic-YES, White Race adults have high rate of HeartDisease. One hot encoder creates new features using feature categories and they have binary values in each feature which helps in identifying direct impact on Target but only issue with oneHotEncoder is it increase the number of features. [OneHotEncoder is not Recommended for High Cardinal Features and Ordinal Features as high cardinality increases Total number of features in the dataset].

3.7.2. Feature Importance

Feature importance is a technique used in machine learning to identify which input variables or features are most relevant for predicting the target variable or outcome. To find features which have strong correlation with our Target and to remove multicollinearity (if present in the dataset).

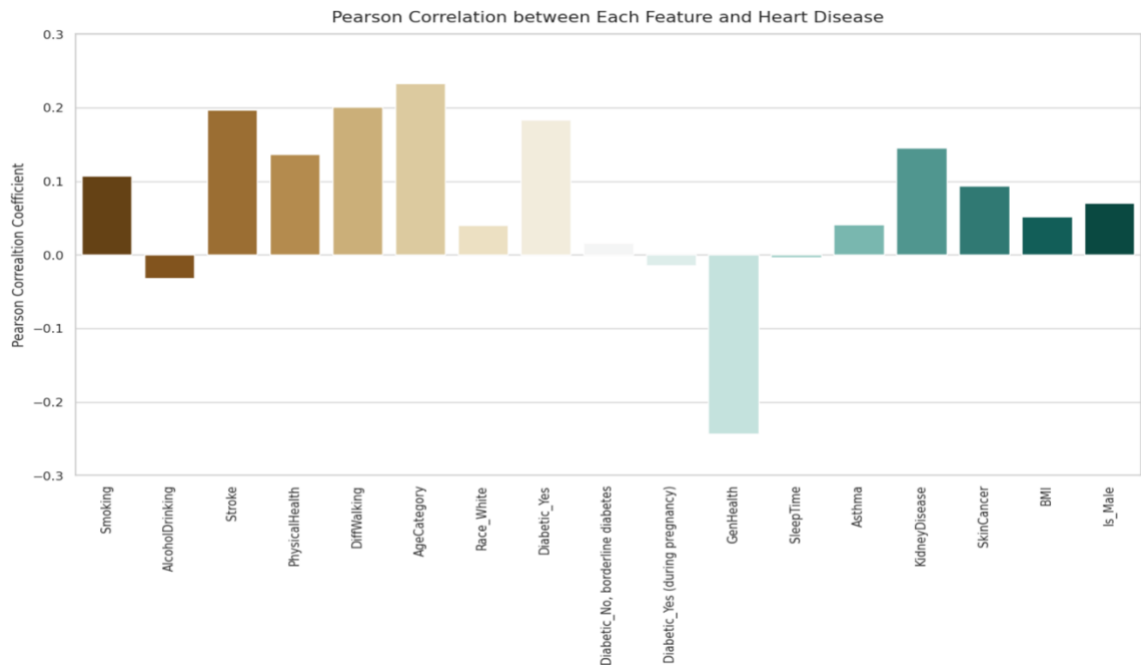
3.7.2.1. Checking for multicollinearity in features



- There seems multicollinearity exists among some of the features like PhysicalHealth, MentalHealth, DiffWalking with GenHealth, as if teh adults are suffering with physical illness or mental illness for many days their genHealth will be poor and adults find with difficult with Walking are not very good in their health condition.
- DiffWalking also has significant negative correlation with PhysicalAcitivity, as expected adults who are regularly doing physical activity wont find any difficulty in walking as they will be active.
- Some of our Race features are correlated with other Race features, so lets consider only Race_white as it has high weightage compare to other race features with Target.
- Diabetic_Yes and Diabetic_No are highly negatively correlated as both are very realted, so lets consider Diabetic_Yes. Diabetic_Yes is again correlated with GenHealth, so lets confirm the multicollinearity using VIF or OLS methods.

3.7.2.2. Feature Importance using Pearson Correlation

- AgeCategory, Stroke, Diabetic_yes have very positive correlation with Target where as AlcholDrinking and GenHealth are very negatively correlated with Target.
- List of Important Features are AgeCategory, Stroke, Diabetic_Yes, KidneyDisease, Smoking, SkinCancer, is_Male, BMI, Asthma, Race_White, Diabetic_No, borderline diabetes, SleepTime', Diabetic_Yes (during pregnancy), AlcoholDrinking, GenHealth



3.7.2.3. OLS Method: Let's check significant features and their weights using OLS Method

- Standard Errors assume that the covariance matrix of the errors is correctly specified.
The smallest eigenvalue is 2.65e-26. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.
- The OLS results confirm the multicollinearity among the features, so we select same features as we selected using Pearson correlation and check for collinearity.
- All the selected features are significant except Diabetic_Yes (during pregnancy).
- List of Important Features are AgeCategory, Stroke, Diabetic_Yes, KidneyDisease, Smoking, SkinCancer, is_Male, BMI, Asthma, Race_White, Diabetic_No, borderline diabetes, SleepTime, AlcoholDrinking, GenHealth

Out [44]: OLS Regression Results

Dep. Variable:	HeartDisease	R-squared:	0.141
Model:	OLS	Adj. R-squared:	0.141
Method:	Least Squares	F-statistic:	2279.
Date:	Tue, 25 Apr 2023	Prob (F-statistic):	0.00
Time:	16:47:08	Log-Likelihood:	-22143.
No. Observations:	319795	AIC:	4.433e+04
Df Residuals:	319771	BIC:	4.459e+04
Df Model:	23		
Covariance Type:	nonrobust		

3.7.2.4. Permutation Importance

- There seems existence of multicollinearity among some of the features like PhysicalHealth, MentalHealth, DiffWalking with GenHealth, as if ten adults are suffering with physical illness or mental illness for many days their genHealth will be poor and adults find with difficult with Walking are not very good in their health condition.
- DiffWalking also has significant negative correlation with PhysicalActivity, as expected adults who are regularly doing physical activity won't find any difficulty in walking as they will be active.
- Some of our Race features are correlated with other Race features, so consider only Race_white as it has high weightage compare to other race features with Target.
- Diabetic_Yes and Diabetic_No are highly negatively correlated as both are very realted, so lets consider Diabetic_Yes. Diabetic_Yes is again correlated with GenHealth, so lets confirm the multicollinearity using VIF or OLS methods.
- Feature Importance of PermutationImportance aligns with above 2 methods (Pearson Correlation weightage and OLS Method), As expected AgeCategory, GenHealth, Stroke, Diabets plays significant role to identify HeartDisease.
- SleepTime, Diabetic During Pregnancy and Borderline Diabetes are not having any significant importance with any of above tried 3 methods, it is better to drop them.
- Final List of Selected Important Features/Variables which are independent of each other are AgeCategory, Stroke, Diabetic_Yes, KidneyDisease, Smoking, SkinCancer, is_Male, BMI, Asthma, Race_White, AlcoholDrinking, GenHealth

3.7.2.5. Variance Inflation Factor (VIF) on selected features

- All the features VIF scores are under 5, so we do not have any multi collinear features.
- AgeCategory, Stroke, Diabetic_Yes, KidneyDisease, Smoking, SkinCancer, is_Male, BMI, Asthma, Race_White, AlcoholDrinking, GenHealth are features are independent of each other and have significance importance in identifying HeartDisease.

3.8. Model Building

3.8.1. Over Sampling:

The data is unbalanced this can lead to biased or inaccurate model performance. The model may tend to predict the majority class more frequently, while the minority class is ignored or misclassified. This can result in a model with high accuracy for the majority class but poor performance for the minority class.

OverSampling: Balancing the each class distribution to build a robust model.

```
In [56]: # OverSampling our Target and creating a new dataframe

class_0 = t_df_imp[t_df_imp['HeartDisease'] == 0]
class_1 = t_df_imp[t_df_imp['HeartDisease'] == 1]

class_1 = class_1.sample(len(class_0),replace=True)
balanced_df = pd.concat([class_0, class_1], axis=0)
print('OverSampled Data Distribution:\n',balanced_df['HeartDisease'].value_counts())

OverSampled Data Distribution:
0    292422
1    292422
Name: HeartDisease, dtype: int64
```

3.8.2. Logistic Regression

The first model we used to is the Logistic Regression model due to its simplicity, interpretability, and effectiveness in many medical diagnoses.

- The model Performance with test sets is Accuracy_score: 76%, Precision_score: 75%, Recall_score: 78% and the F1-score: 77%

- Yes!, all the important features in Test and Validation sets are following same distribution as Train set.

```
In [60]: # lets build a Logistic Model and check the model performance with Test and Validation Sets.
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train)
lr_y_predict = lr.predict(X_test)

print("<----- Model Performance with Test set ----->")
print(f'model: {str(lr)}')
print(f'Accuracy_score: {accuracy_score(y_test,lr_y_predict)}')
print(f'Precision_score: {precision_score(y_test,lr_y_predict)}')
print(f'Recall_score: {recall_score(y_test,lr_y_predict)}')
print(f'F1-score: {f1_score(y_test,lr_y_predict)}')

cm = confusion_matrix(y_test,lr_y_predict)
plt.figure(figsize=(10,5))
ax = sns.heatmap(cm/np.sum(cm),fmt='.2%', annot=True, cmap='Blues')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['No HeartDisease','HeartDisease'])
ax.yaxis.set_ticklabels(['No HeartDisease','HeartDisease'])
plt.show()

<----- Model Performance with Test set ----->
model: LogisticRegression(random_state=0)
Accuracy_score: 0.7619547003773068
Precision_score: 0.75
Recall_score: 0.779747533972546
F1-score: 0.7645845310967566
```

- Model Performance with Validation set is Accuracy_score: 76%, Precision_score: 75%, Recall_score: 78%, F1-score: 77%
- Model AUC score for training, test, and Validation data are 83%,83%, and 83% respectively.

```

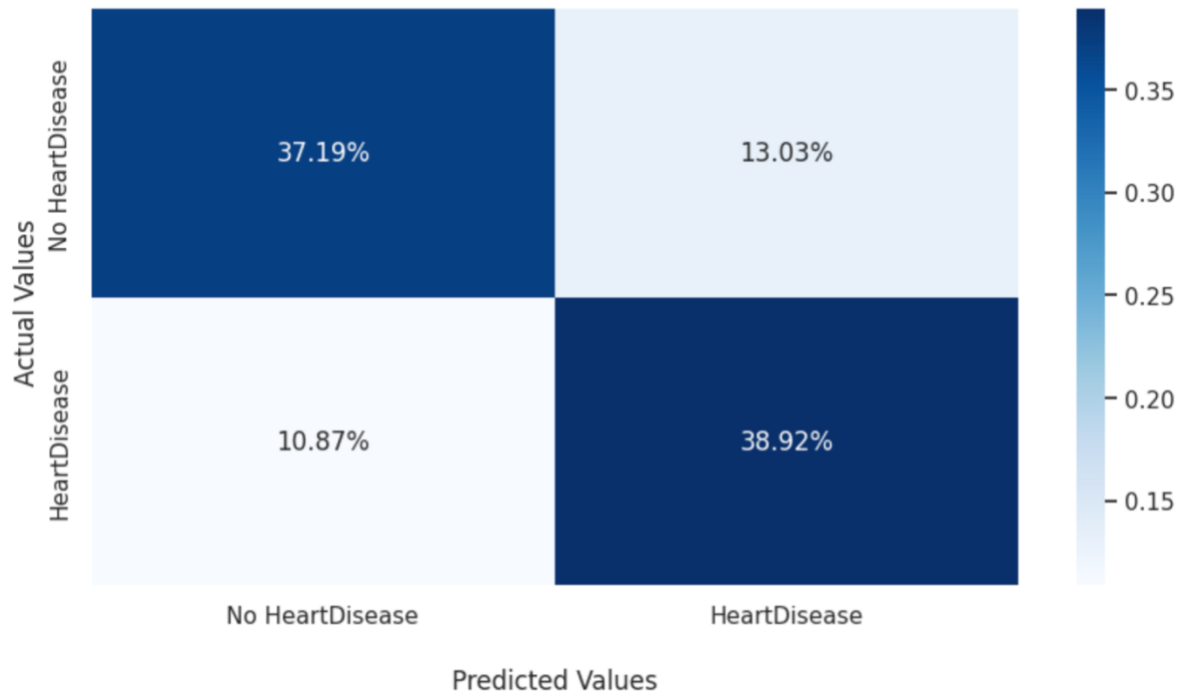
<----- Model Performance with Validation set ----->
model: LogisticRegression(random_state=0)
Accuracy_score: 0.761043611200515
Precision_score: 0.7492351187018318
Recall_score: 0.7816565656565656
F1-score: 0.7651025291174783

```

```

-----
Model AUC Score on Training Data: 0.840198414422887
Model AUC Score on Test Data: 0.8387856212117553
Model AUC Score on Validation Data: 0.8384597153329627

```



3.8.3. Random Forest

- Model Performance is very stable across all the sets (Train, Test and Validation sets).
- This is the best model so far compared to Logistic as we can see the RandomForest model AUC score is ~85% whereas Logistic model AUC score is just 83%.
- Moreover RandomForest model performance is pretty stable compared to Logistic model on every dataset like (Train, Test and Validation).

```
Model Score on Training Data: 0.7706798091214895
Model Score on Test Data: 0.7682013519213013
Model Score on Validation Data: 0.7661731573865465
-----
Model AUC Score on Training Data: 0.8495585317123676
Model AUC Score on Test Data: 0.8466466529846044
Model AUC Score on Validation Data: 0.8449129812668792
```

Model Performance is very stable across all the sets (Train, Test and Validation sets)

- This is the best model so far compare to Logistic as we can see the RandomForest model AUC score is ~85% where as Logistic model AUC score is
- Moreover RandomForest model performance is pretty stable compare to Logistic model on every dataset like (Train, Test and Validation).
- Based on the above performance, selecting the RandomForestClassifier Model.

```
[66]: #####Tuning Random forest with different parameters
forest2 = RandomForestClassifier(n_estimators=50, max_depth=20, max_features='sqrt', criterion='entropy', random_state=0)
forest2.fit(X_train,y_train)
print(f"Model Score on Training Data: {forest2.score(X_train,y_train)}")
print(f"Model Score on Test Data: {forest2.score(X_test,y_test)}")
print(f"Model Score on Validation Data: {forest2.score(X_val, y_val)}")
print("-----")
print(f"Model AUC Score on Training Data: {roc_auc_score(y_train, forest2.predict_proba(X_train)[:,-1])}")
print(f"Model AUC Score on Test Data: {roc_auc_score(y_test, forest2.predict_proba(X_test)[:,-1])}")
print(f"Model AUC Score on Validation Data: {roc_auc_score(y_val, forest2.predict_proba(X_val)[:,-1])}")

Model Score on Training Data: 0.8140543802594964
Model Score on Test Data: 0.8018056014681911
Model Score on Validation Data: 0.8004103636948825
-----
Model AUC Score on Training Data: 0.8995244493825688
Model AUC Score on Test Data: 0.8836961527913156
Model AUC Score on Validation Data: 0.8833437815378364
```

Based on the performance, selecting the Random Forest Classifier Model.

3.9. Model Interpretability using SHAP

- The Shap values indicate, with increase in Age, the chances of getting heart disease are also High compared to younger people.
- General Health shows very realistic trend that if someone's general health is poor/bad, they are high prone to Heart Disease
- Male_Gender, Smoking, BMI, Asthma follows the same trend, if they are Male / have smoking habit / has high BMI value / has Asthma problem, they are highly prone to diagnosed with heart disease.
- Stroke & Kidney Disease shows realistic trend that If someone suffered with Stroke / Kidney Disease in the past have high chances of getting Heart Disease.
- It is interesting behavior about Alcohol Drinking habit, irrespective of whether one drinks or not, cannot escape from diagnosed with heart disease.
- As we inferred from model feature importance, if someone is facing critical Health issues then they are highly prone to heart disease.

4. Technical Specifications

4.1. Airflow

Airflow is a powerful open-source platform designed for creating, scheduling, and monitoring workflows. It is a popular choice for data engineers and data scientists who need to manage complex data pipelines with multiple dependencies, as it provides a flexible and scalable framework for building, testing, and deploying these pipelines.

The data pipeline for Pulse Vision includes the following steps –

1. Data extraction
2. Data ingestion
3. Data cleaning
4. Data storage

Directed Acyclic Graphs or DAGs are used to execute the collection of tasks. This DAG is scheduled to extract and updated the latest data from CDC to provide accurate predictions. It is currently scheduled to run every year as the data present on CDC website is annual. The following tasks are performed to implement the data pipeline:

1. task_directory_cleanup –

This task is used to clean the working directory in airflow to start the process afresh and avoid any conflicts because of previous failed tasks, if any.

2. task_download_files –

This task retrieves the most recent data from the CDC and transfer it to the S3 bucket. The data is compressed in a zip file format and must be decompressed to obtain an XPT file, which is also kept on the same S3 bucket.

3. task_data_cleanup –

This task converts the XPT file to CSV which contains unstructured and unclean data. We then identify key features from this dataset and clean the data and save it in CSV format.

4. task_upload_to_s3 –

This task uploads the structured and cleaned data file to s3 which is then utilized to train our prediction model.

The airflow instance for this project is hosted on cloud and can be accessed using the following link: <http://23.21.117.161:8080/home>

Use the following credentials for access –

Username : pulsevision

Password : team11

To implement this pipeline on your own machine, follow these steps:

Pre-requisites

- Code Editor (PyCharm, VS Code, etc)
- Docker
- Python v3.2 or later

Steps on run Airflow locally

1. Clone the airflow branch using the following code on terminal -

```
git clone --branch airflow https://github.com/BigDataIA-Spring2023-Team-11/PulseVision.git
```

2. Move to the project directory and run the following command in terminal to create a .env file

```
nano .env
```

3. Add the following environment variables with values:

```
AIRFLOW_UID=501  
AIRFLOW_PROJ_DIR=./airflow  
ACCESS_KEY=  
SECRET_KEY=  
SOURCE_BUCKET=  
API_KEY=  
KMP_DUPLICATE_LIB_OK=TRUE
```

4. Save file while exiting the editor -> control + x

5. Install requirements using command -

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

6. Ensure docker is running in background. Run following commands -

```
docker-compose build  
docker compose up airflow-init  
docker compose up
```

7. Once the localhost URL is shared, go to : <https://localhost:8080/>

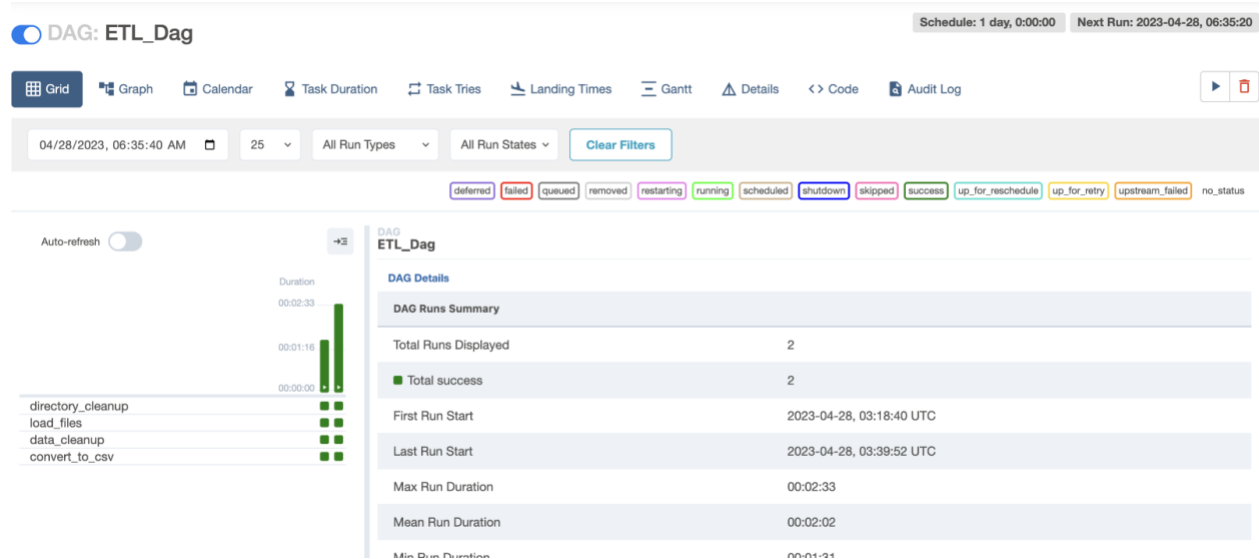
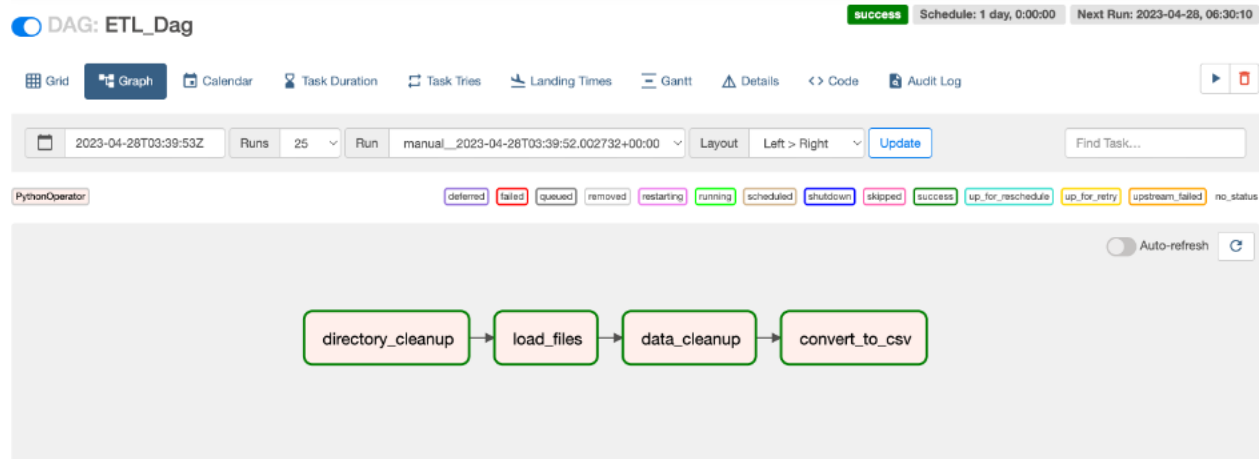
8. Login using following credentials -

```
Username: a4@team11  
Password: team11
```

9. Review DAG: ETL_dag

10. Go to dags to trigger run, view logs, graphs etc.

The 'ETL_dag' as shown in Airflow –

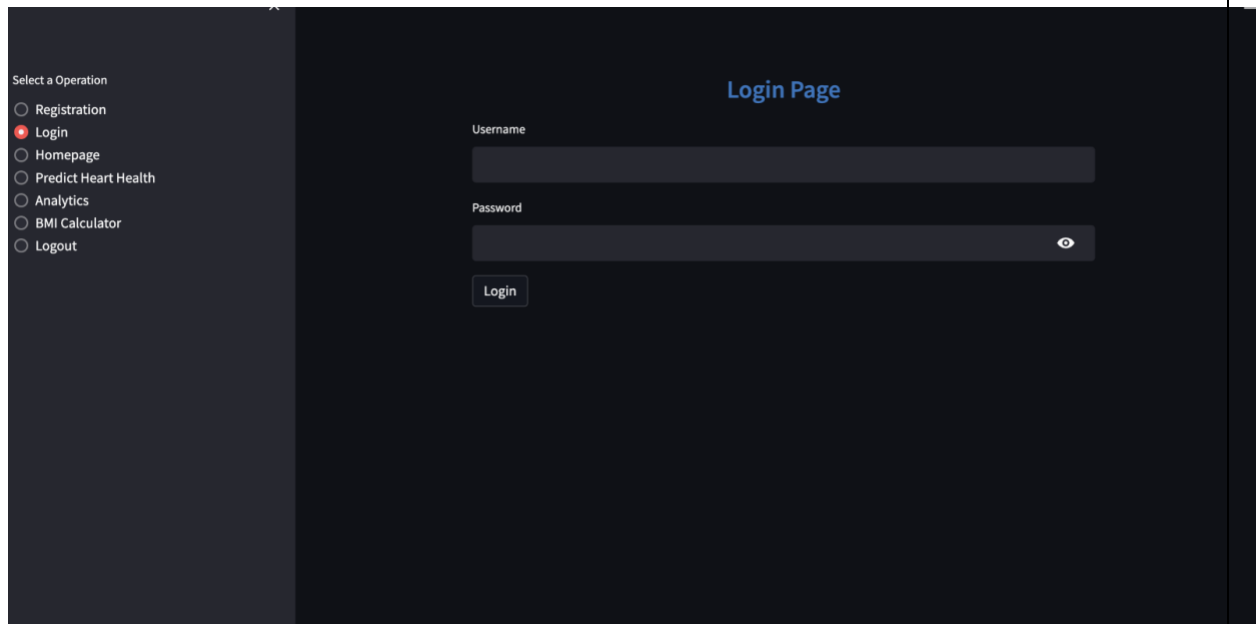


4.2.Streamlit

Streamlit is an open-source Python library for building interactive web applications.Our frontend UI is designed with the help of this easy-to-use python library.

4.2.1. Login Page

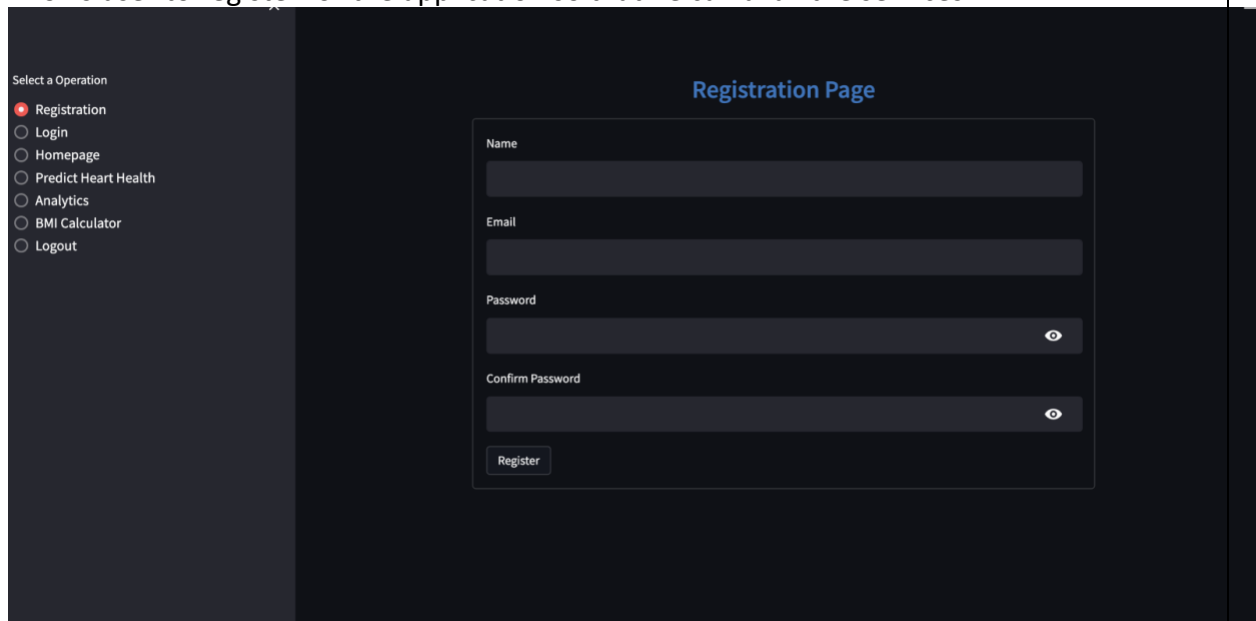
The Login page allows the user to enter his credentials and login to the application.



The screenshot displays the 'Login Page' interface. On the left, a sidebar titled 'Select a Operation' contains a list of radio buttons: 'Registration', 'Login' (which is selected and highlighted with a red dot), 'Homepage', 'Predict Heart Health', 'Analytics', 'BMI Calculator', and 'Logout'. The main content area on the right is titled 'Login Page' in blue. It features two input fields: 'Username' and 'Password'. The 'Password' field includes a toggle icon (an eye) to switch between visible and hidden states. Below the input fields is a 'Login' button.

4.2.2. User Registration Page

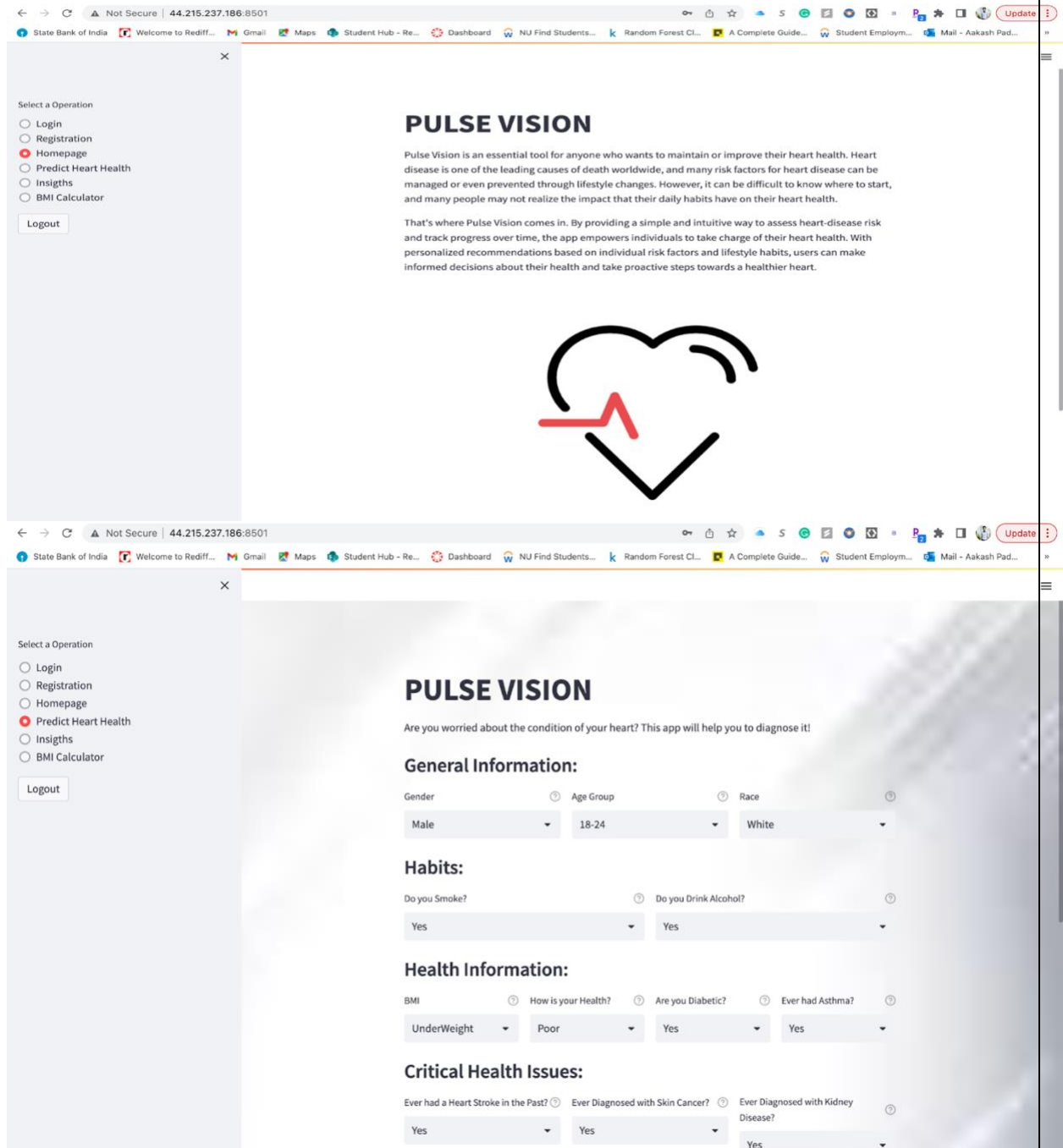
Allows user to register for the application so that he can avail the services.



The screenshot displays the 'Registration Page' interface. The sidebar on the left is identical to the login page, with 'Registration' selected. The main content area is titled 'Registration Page' in blue. It contains four input fields: 'Name', 'Email', 'Password', and 'Confirm Password'. Both the 'Password' and 'Confirm Password' fields have toggle icons (eyes) to manage visibility. A 'Register' button is positioned at the bottom of the form.

4.2.3. Prediction Page

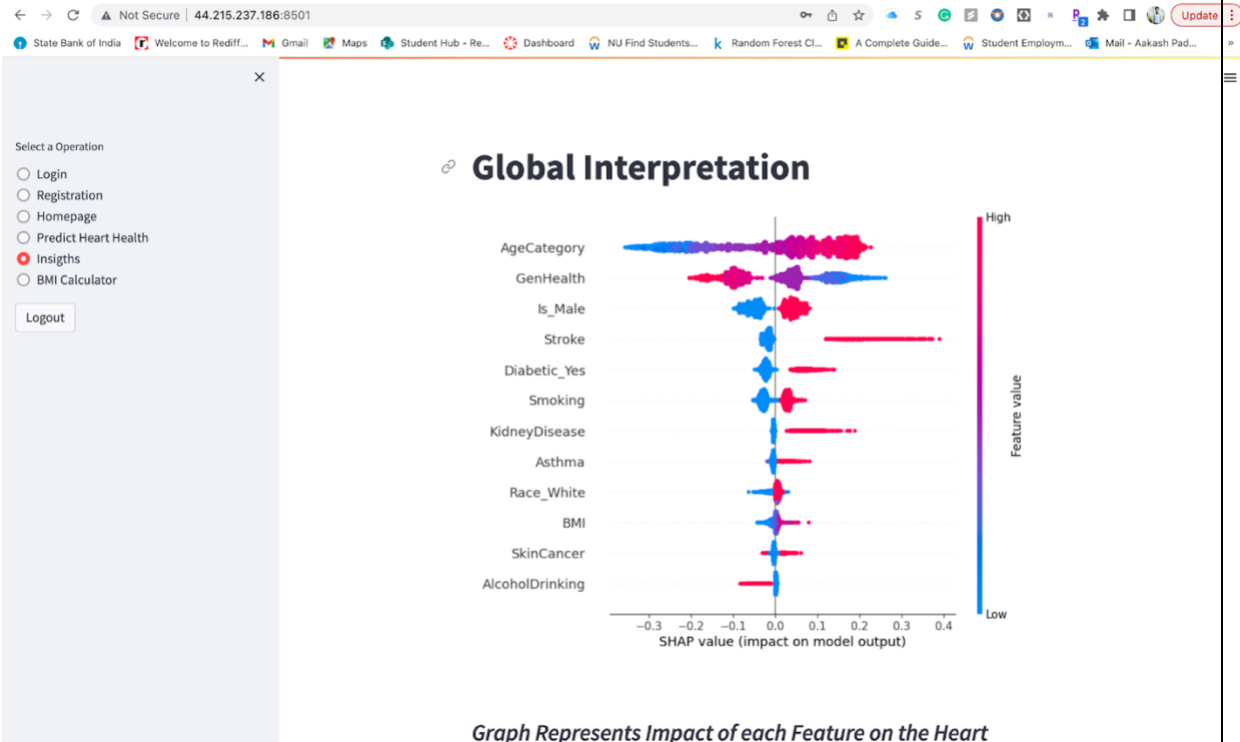
This page takes in a bunch of inputs from the user and gives the predicted risk for heart disease. It also allows the user to change the parameters and see how it would affect the risk percentage.



The screenshot displays the 'PULSE VISION' prediction interface. On the left, a sidebar menu titled 'Select a Operation' includes options for Login, Registration, Homepage (selected), Predict Heart Health, Insights, BMI Calculator, and a Logout button. The main content area features the 'PULSE VISION' title, a descriptive paragraph about heart disease risk, and a heart icon with a red ECG line. Below this, a form titled 'PULSE VISION' asks, 'Are you worried about the condition of your heart? This app will help you to diagnose it!'. The form is organized into four sections: 'General Information' with dropdowns for Gender (Male), Age Group (18-24), and Race (White); 'Habits' with dropdowns for 'Do you Smoke?' (Yes) and 'Do you Drink Alcohol?' (Yes); 'Health Information' with dropdowns for BMI (UnderWeight), 'How is your Health?' (Poor), 'Are you Diabetic?' (Yes), and 'Ever had Asthma?' (Yes); and 'Critical Health Issues' with dropdowns for 'Ever had a Heart Stroke in the Past?' (Yes), 'Ever Diagnosed with Skin Cancer?' (Yes), and 'Ever Diagnosed with Kidney Disease?' (Yes). The browser's address bar shows the URL '44.215.237.186:8501'.

4.2.4. Analytics Page

This page allows the user to view its analytic dashboard.



BMI Calculator:

BMI Calculator

Enter your weight (in kg)
0.00

Enter your height (in cm)
0.00

Calculate BMI

Made with Streamlit

4.3. Fast API

FastAPI is a modern, fast (hence the name), web framework for building APIs with Python. It is designed to be easy to use, with a focus on developer productivity, code readability, and maintainability. FastAPI leverages Python's type annotations to provide automatic data validation and documentation for APIs. It also supports asynchronous programming using Python's asyncio library, which allows for fast and efficient handling of requests and responses.

In our case, the Fast API is used to trigger the predictions from frontend. When the user enters in the input on the prediction page and hits the Predict Button, the API is hit to take these inputs and give the predictions.

1. Install FastAPI: Install FastAPI using the command “ pip install fastapi”
2. Create a FastAPI app: Create a Python file and import FastAPI using the following code:
from fastapi import FastAPI
app = FastAPI()
4. Define User Input:

```
class UserInput(BaseModel):  
    AgeCategory: int  
    Stroke: int  
    Diabetic_Yes: int  
    KidneyDisease: int  
    Smoking: int  
    SkinCancer: int  
    Is_Male: int  
    BMI: int  
    Asthma: int  
    Race_White: int  
    AlcoholDrinking: int  
    GenHealth: int
```

3. Define endpoints:

```
@app.post('/predict', status_code=status.HTTP_200_OK)
async def predict(userinput: UserInput) -> dict:
    try:
        # read the input dict into json and then into df
        df = pd.DataFrame([json.loads(userinput.json())])
        # predict
        # prediction = model.predict(df)
        prob = model.predict_proba(df)
        prediction = model.predict(df)
        print(f"{prediction[0]} -----000000")
        # prob = model.predict_proba(df)
```

4. Defining the response for the end point:

```
response = {}
# # # first prediction value rounded off
# response['value_0'] = (prob[0][0] * 100 ).round(0)
# # second prediction value rounded off
response['prediction_probability'] = (prob[0][1] * 100 ).round(0)
response['prediction'] = int(prediction[0])
return response

# prediction = log_model.predict(df)
# prediction_prob = log_model.predict_proba(df)
# pred_prob = (100 * prob[0][1]).round(0)
# return {"prediction": prediction, "prediction_probability": pred_prob}

except:
    raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Cannot Predict, Try again")
```

5. Run the app : uvicorn api_main:app --reload

6. Test the endpoints:

Heart Disease Prediction 0.1.0 OAS3

/openapi.json

default

POST /get_predict Predict

4.4. Pytest

Pytest is a popular Python testing framework that can be used for testing APIs. The general steps for using Pytest for API testing:

1. Install pytest and the necessary testing libraries, such as requests and jsonschema, using pip.
2. Create a separate directory for your API tests and create a Python file to write your tests.
3. Write test functions using the pytest syntax and use the requests library to make HTTP requests to your API endpoints.
4. Use assertions to check the response status codes, headers, and content.

To run the file type the command in your terminal:

Pytest test_api.py

The following are the snap shots of the pytests that we conducted:

```
new *  
def test_server_response():  
    response = requests.get('http://localhost:8000/docs')  
    assert response.status_code == 200  
new *
```

```
new *
def test_predict_endpoint():
    url = 'http://localhost:8000/predict'
    headers = {'Content-Type': 'application/json'}
    data = {
        "AgeCategory": 1,
        "Stroke": 1,
        "Diabetic_Yes": 1,
        "KidneyDisease": 0,
        "Smoking": 1,
        "SkinCancer": 0,
        "Is_Male": 1,
        "BMI": 1,
        "Asthma": 0,
        "Race_White": 1,
        "AlcoholDrinking": 0,
        "GenHealth": 1
    }
    response = requests.post(url, json=data, headers=headers)
    assert response.status_code == 200
    assert 'value_0' in response.json()
    assert 'value_1' in response.json()
```

```
def test_predict_endpoint_422():
    url = 'http://localhost:8000/predict'
    headers = {'Content-Type': 'application/json'}
    # Invalid data - missing required field
    data = {
        "Stroke": 1,
        "Diabetic_Yes": 1,
        "KidneyDisease": 0,
        "Smoking": 1,
        "SkinCancer": 0,
        "Is_Male": 1,
        "BMI": 1,
        "Asthma": 0,
        "Race_White": 1,
        "AlcoholDrinking": 0,
        "GenHealth": 1
    }
    response = requests.post(url, json=data, headers=headers)
    assert response.status_code == 422
```

Output:

```
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 5 passed, 2 warnings in 4.81s =====
(venv) (base) akshaysawant@akshays-MacBook-Pro api_utils %
```

4.5. Docker

Docker has been used for containerization of the application to run the applications in a portable and consistent manner across different environments.

4.6. AWS Services

We are using various Aws services such as:

1. S3 buckets to store the cleaned data from the CDC website and to host the great expectation reports.
2. The Airflow is hosted on EC2 instance.
3. CloudWatch to log user activity.

4.7. Git Actions

1. Integration testing of our data pipelines as part of your CI workflows
 - This Action allows to trigger a Great Expectation *Checkpoint*, which runs a suite of data tests against your data in GitHub and links directly to the validation report (Data Docs).
 - This means we can test whether any changes to our data pipelines had unintended consequences.
 - The Great Expectations Action does not run your data pipelines. This will need to be configured as a separate GitHub workflow.
2. Integrating testing with Git Actions to automatically run our tests on every push or pull request to your repository
 1. Create a new workflow file in your repository's **.github/workflows** directory. The file name should end with **.yml**, for example **pytest.yml**.
 2. Define the workflow using the YAML syntax. The workflow should include the following steps:
 - Checkout the repository code
 - Install the necessary dependencies, including Pytest and any libraries required for your tests.
 - Run the Pytest command to execute your tests.

CONCLUSION

In conclusion, the heart risk prediction project using streamlit UI and random forest classifier, combined with cutting-edge technologies such as Airflow, Git Actions, Docker, Great Expectations, and FastAPI, has proven to be a powerful and effective tool for predicting heart disease risk. The random forest classifier performed well in predicting heart disease risk with an accuracy score of 84%, and the use of streamlit UI made it easy to deploy and interact with the model.

Additionally, the integration of Airflow, Git Actions, and Docker has provided a seamless pipeline for model deployment and version control. Great Expectations helped ensure data quality and integrity, while FastAPI enabled the development of an API for easy integration with other applications.

Overall, the project highlights the importance of using modern technologies and tools to develop robust and reliable machine learning applications. With its high accuracy and user-friendly interface, the heart risk prediction project has the potential to make a significant impact in the field of healthcare and improve patient outcomes.

References:

1. Scikit-learn. (n.d.). Scikit-learn: Machine learning in Python. Retrieved from <https://scikit-learn.org/stable/documentation.html>
2. Author, A. A. (Year, Month Day). Title of article. Medium. Retrieved from <https://medium.com/>
3. Author, A. A. (Year, Month Day). Title of article. Analytics Vidhya. Retrieved from <https://analyticsvidhya.com/>
4. Author, A. A. (Year, Month Day). Title of article. Towards Data Science. Retrieved from <https://towardsdatascience.com/>
5. Author, A. A. (Year, Month Day). Title of notebook. Kaggle. Retrieved from <https://kaggle.com/>
6. Aiskunks. (n.d.). GLM H2O AutoML Demo. GitHub. Retrieved from https://github.com/aiskunks/Skunks_Skool/blob/main/H2O_AutoML_IPYNB/glm_h2oworld_demo.ipynb
7. Aiskunks. (n.d.). GLRM Census Labor Violations Demo. GitHub. Retrieved from https://github.com/aiskunks/Skunks_Skool/blob/main/H2O_AutoML_IPYNB/glrm.census.labor.violations.ipynb
8. Aiskunks. (n.d.). GLRM Walking Gait Demo. GitHub. Retrieved from https://github.com/aiskunks/Skunks_Skool/blob/main/H2O_AutoML_IPYNB/glrm.walking.gait.ipynb
9. Sahu, S. (2020, April 13). Understanding Feature Scaling in Machine Learning. Analytics Vidhya. Retrieved from <https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/>
10. Aiskunks. (n.d.). I2SL. GitHub. Retrieved from https://github.com/aiskunks/Skunks_Skool/tree/main/I2SL
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830. Retrieved from https://scikit-learn.org/stable/modules/permutation_importance.html
12. Scikit-learn. (n.d.). Permutation Importance. Scikit-learn: Machine learning in Python. Retrieved from https://scikit-learn.org/stable/modules/permutation_importance.html
13. Scikit-learn. (n.d.). Logistic Regression. Scikit-learn: Machine learning in Python. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
14. Aiskunks. (n.d.). Logistic Regression Using Python and Excel. GitHub. Retrieved from https://github.com/aiskunks/Skunks_Skool/blob/main/Logistic%20Regression%20Using%20Python%20and%20Excel.ipynb
15. Kaggle. (n.d.). Kaggle. Retrieved from <https://kaggle.com/>
16. Sharma, N. (2021, May 17). Detecting and Treating Outliers: Treating the Odd One Out. Analytics Vidhya. Retrieved from

<https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/>

17. Sharma, N. (2021, November 22). Model Explainability: Demystifying the Black-Box Models. Analytics Vidhya. Retrieved from <https://www.analyticsvidhya.com/blog/2021/11/model-explainability/>