**Project Name :AI-Based Diabetes Prediction System**

**Definition:**A system is used to predict whether a patient has diabetes based on some of its health- related details such as BMI (Body Mass Index), blood pressure, Insulin.etc

**Abstract:**

Diabetes mellitus is a chronic health condition affecting millions of individuals worldwide. Early detection and management of diabetes are crucial for preventing complications and improving patient outcomes. This project introduces an AI-based Diabetes Prediction System, which leverages machine learning techniques to predict the risk of diabetes in individuals.

**Module 1: Data Collection and Preprocessing**

* In this module, relevant health data is collected, including medical history, age, gender, family history, and lifestyle factors.
* Data preprocessing techniques such as cleaning, normalization, and feature selection are applied to ensure the data's quality and relevance.

**Module 2: Feature Engineering**

* Feature engineering is employed to create meaningful features from the collected data, enhancing the model's predictive capabilities.
* This module involves techniques like one-hot encoding, feature scaling, and the creation of composite features.

**Module 3: Machine Learning Model Development**

* Various machine learning algorithms, such as logistic regression, decision trees, and support vector machines, are implemented to build the predictive model.
* The model is trained on a labeled dataset, and hyperparameter tuning is performed to optimize its performance.

**Module 4: Evaluation and Validation**

* The model's performance is assessed using metrics such as accuracy, precision, recall, and F1-score.
* Cross-validation techniques are employed to ensure the model's robustness and generalizability.

**Module 5: User Interface (UI) Development**

* A user-friendly interface is created to allow users to input their health information and obtain predictions.
* The UI provides clear explanations of the predictions and offers insights into risk factors.

**Module 6: Integration with Healthcare Systems**

* The system is integrated with electronic health record (EHR) systems to access and update patient data seamlessly.
* Privacy and security measures are implemented to protect patient information.

**Module 7: Continuous Learning and Improvement**

* The model is continuously updated with new data to adapt to changing healthcare trends.
* Feedback mechanisms are established to gather user input for further enhancements.

**Data analysis:**

Here one will get to know about how the data analysis part is done in a data science life cycle.

**Exploratory data analysis:**

EDA is one of the most important steps in the data science project life cycle and here one will need to know that how to make inferences from the visualizations and data analysis

**Model building:**

Here we will be using 4 ML models and then we will choose the best performing model.

**Saving model:**

Saving the best model using pickle to make the prediction from real data.

# Importing libraries

Import numpy as np Import pandas as pd

Import matplotlib.pyplot as plt

Import seaborn as sns

Sns.set()

From mlxtend.plotting import plot\_decision\_regions Import missingno as msno

From pandas.plotting import scatter\_matrix

From sklearn.preprocessing import StandardScaler From sklearn.model\_selection import train\_test\_split From sklearn.neighbors import KNeighborsClassifier

From sklearn.metrics import confusion\_matrix From sklearn import metrics

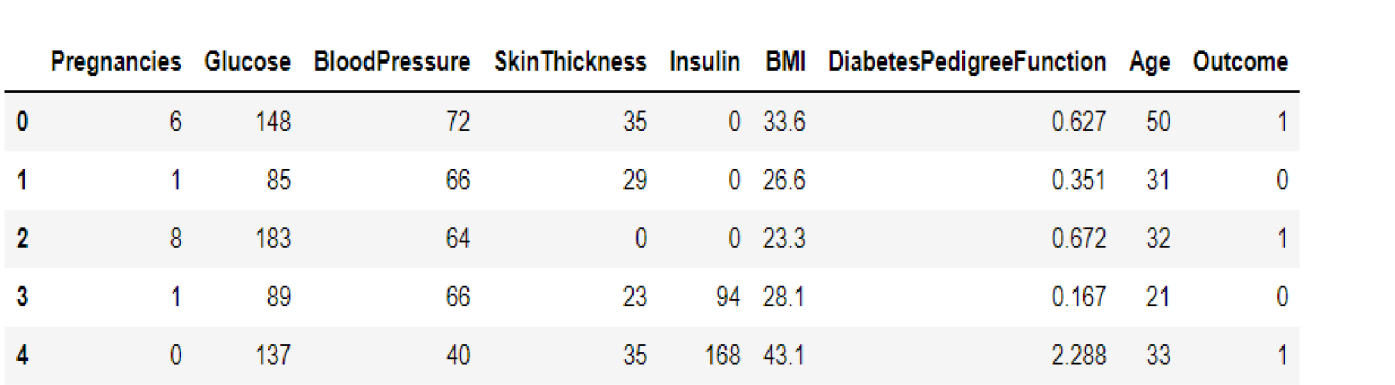
From sklearn.metrics import classification\_report Import warnings

warnings.filterwarnings(‘ignore’)

%matplotlib inline

**Here we will be reading the dataset which is in the CSV format**

diabetes\_df = pd.read\_csv(‘diabetes.csv’) diabetes\_df.head()

**Output:**

# Exploratory Data Analysis (EDA)

Now let’ see that what are columns available in our dataset.

diabetes\_df.columns

Output:

Index([‘Pregnancies’, ‘Glucose’, ‘BloodPressure’, ‘SkinThickness’, ‘Insulin’, ‘BMI’, ‘DiabetesPedigreeFunction’, ‘Age’, ‘Outcome’],

Dtype=’object’)

Information about the dataset

diabetes\_df.info()

Output:

RangeIndex: 768 entries, 0 to 767 Data columns (total 9 columns):

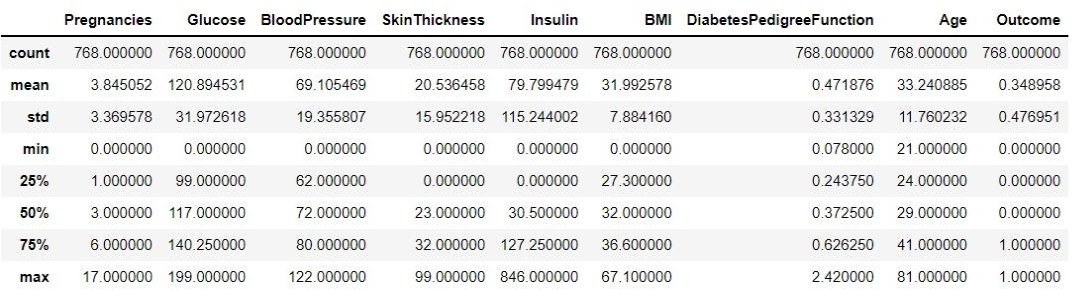
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # |  | Column | Non-Null Count |  | Dtype |
| 0 |  | Pregnancies | 768 non-null |  | int64 |
| 1 |  | Glucose | 768 non-null |  | int64 |
| 2 |  | BloodPressure | 768 non-null |  | int64 |
| 3 |  | SkinThickness | 768 non-null |  | int64 |
| 4 |  | Insulin | 768 non-null |  | int64 |
| 5 |  | BMI | 768 non-null |  | float64 |
| 6 |  | DiabetesPedigreeFunction | 768 non-null |  | float64 |
| 7 |  | Age | 768 non-null |  | int64 |
| 8 |  | Outcome | 768 non-null |  | int64 |

Dtypes: float64(2), int64(7) Memory usage: 54.1 KB

To know more about the dataset

Diabetes\_df.describe()

**Output:**



To know more about the dataset with transpose – here T is for the transpose

diabetes\_df.describe().T

**Output:**



Now let’s check that if our dataset have null values or not

diabetes\_df.isnull().head(10)

**Output:**



**Now let’s check the number of null values our dataset has.**

diabetes\_df.isnull().sum()

##### Output:

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

DiabetesPedigreeFunction 0

Age 0

Outcome 0

dtype: int64

Here from the above code we first checked that is there any null values from the IsNull() function then we are going to take the sum of all those missing values from

the sum() function and the inference we now get is that there are no missing values but that is actually not a true story as in this particular dataset all the missing values were given the 0 as a value which is not good for the authenticity of the dataset.

Hence we will first replace the 0 value with the NAN value then start the imputation process.

Diabetes\_df\_copy = diabetes\_df.copy(deep = True) Diabetes\_df\_copy[[‘Glucose’,’BloodPressure’,’SkinThickness’,’Insulin’,’BMI’]] = diabetes\_df\_copy[[‘Glucose’,’BloodPressure’,’SkinThickness’,’Insulin’,’BMI’]].replace( 0,n p.NaN)

**# Showing the Count of NANs**

Print(diabetes\_df\_copy.isnull().sum())

|  |  |
| --- | --- |
| **Output:** |  |
| Pregnancies | 0 |
| Glucose | 5 |
| BloodPressure | 35 |
| SkinThickness | 227 |
| Insulin | 374 |
| BMI | 11 |
| DiabetesPedigreeFunction | 0 |
| Age | 0 |
| Outcome | 0 |
| dtype: int64 |  |

As mentioned above that now we will be replacing the zeros with the NAN values so that we can impute it later to maintain the authenticity of the dataset as well as trying to have a better Imputation approach i.e to apply mean values of each column to the null values of the respective columns.

* + **Project Source Code and Result**

**1 AI Diabetes Prediction using Python + Pandas:**

Diabetes is a chronic (long-lasting) health condition that affects how your body turns food into energy. Most of the food you eat is broken down into sugar (also called glucose) and released into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin.

###### Table Content

Introduction Installing Libraries Importing Data

Missing Value Analysis Exploratory Data Analysis Feature Engineering Modeling Hyperparameter Tuning Prediction

###### Objectives

To experiment with different classification methods to see which yields the highest accuracy

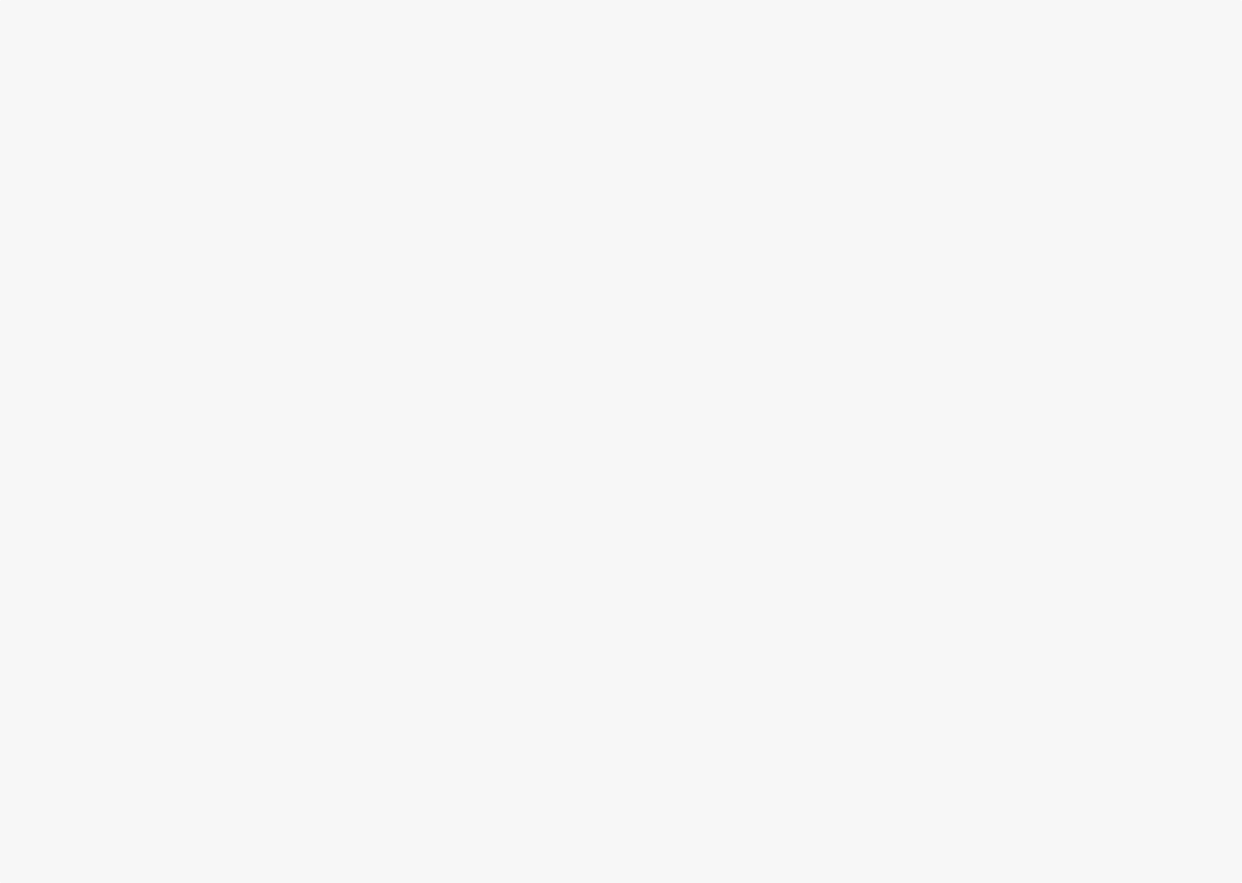
Classify whether someone has diabetes or not from given features To determine which features are the most indicative of diabetes

###### Dataset

I have used [Pima Indians Diabetes Database](https://www.kaggle.com/uciml/pima-indians-diabetes-database) Kaggle Dataset

The dataset contains below features and labels: 1. Pregnancies 2. Glucose 3. BloodPressure 4. SkinThickness 5. Insulin 6. BMI 7. DiabetesPedigreeFunction 8. Age 9. Outcome

1. Installing Libraries

[ ]: **import numpy as np** *# linear algebra*

**import pandas as pd** *# data processing, CSV file I/O (e.g. pd.read\_csv)* **import seaborn as sns** *# for data visualization*

**import matplotlib.pyplot as plt** *# to plot charts* **from collections import** Counter

**import os**

*# Modeling*

**from sklearn.preprocessing import** QuantileTransformer

**from sklearn.metrics import** confusion\_matrix, accuracy\_score, precision\_score **from sklearn.ensemble import** RandomForestClassifier, AdaBoostClassifier,␣

↪GradientBoostingClassifier, VotingClassifier **from sklearn.linear\_model import** LogisticRegression **from sklearn.neighbors import** KNeighborsClassifier **from sklearn.tree import** DecisionTreeClassifier

**from sklearn.svm import** SVC

**from sklearn.model\_selection import** GridSearchCV, cross\_val\_score,␣

↪StratifiedKFold, learning\_curve, train\_test\_split

*# Directory Structure*

**for** dirname, \_, filenames **in** os.walk('/kaggle/input'):

**for** filename **in** filenames: print(os.path.join(dirname, filename))

I have imported most common libraries used in python for machine learning such as Pandas, Seaborn, Matplitlib etc

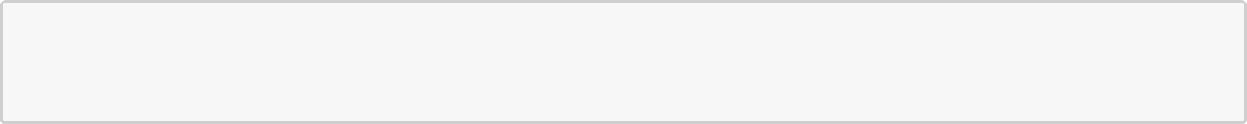
###### Importing Data

[ ]:



df = pd.read\_csv("../input/pima-indians-diabetes-database/diabetes.csv")

[ ]:



*# Get familier with dataset structure*

df.info()

Excepting BMI and DiabetesPedigreeFunction all the columns are integer. Outcome is the label containing 1 and 0 values. 1 means person has diabetes and 0 mean person is not diabetic

[ ]:

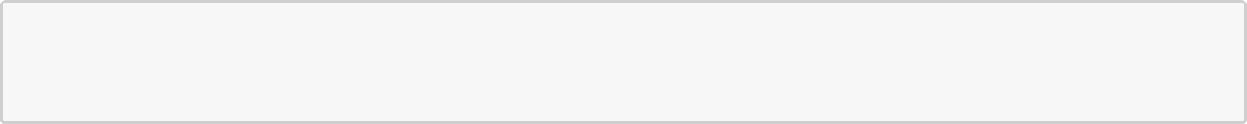


df.describe()

There are 768 records in the dataset, in which mean age of people is 33

[ ]:

[ ]:



*# Explore missing values*

df.isnull().sum()

1. Missing Value Analysis

Next, i will cleanup the dataset which is the important part of data science. Missing data can lead to wrong statistics during modeling and predictions.



df.describe()

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

I observed that there is no missing values in dataset however the features like Glucose, BloodPres- sure, Insulin, SkinThickness has 0 values which is not possible. We have to replace 0 values with either mean or median values of specific column



df['Glucose'] = df['Glucose'].replace(0,df['Glucose'].mean()) df.Glucose.value\_counts()



*# Correcting missing values in blood pressure*

df[df['BloodPressure'] == 0]['BloodPressure'].value\_counts() df['BloodPressure'] = df['BloodPressure'].replace(0,df['BloodPressure'].mean())

There are 35 records with 0 BloodPressure in dataset

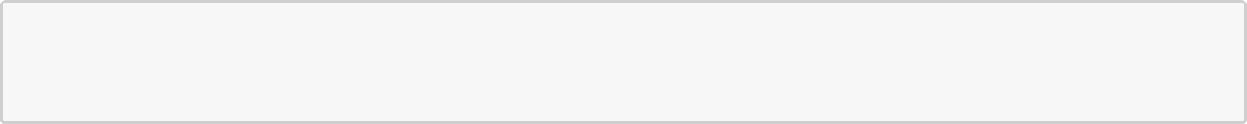


*# Correcting missing values in BMI*

df[df['BMI'] == 0]['BMI'].value\_counts()

df['BMI'] = df['BMI'].replace(0, df['BMI'].median())

Now i have dataset without missing values in features which is good



*# Correct missing values in Insulin and SkinThickness*

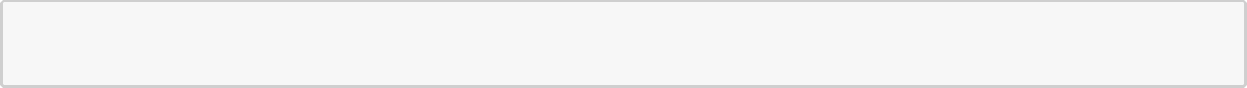
df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness']. ↪median())

df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())

*# Review dataset statistics*

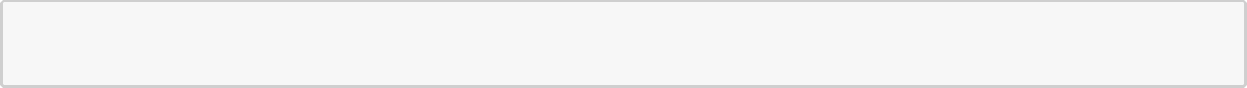
df.describe()

###### Exploratory Data Analysis

[ ]: *# Show top 5 rows*

df.head()

**Correlation**

[ ]: plt.figure(figsize=(13,10))

sns.heatmap(df.corr(),annot=**True**, fmt = ".2f", cmap = "coolwarm")

According to observation, features like Pregnancies, Gluecose, BMI, and Age is more correlated with Outcome

**Pregnancies**

[ ]: *# Explore Pregnancies vs Outcome*

plt.figure(figsize=(13,6))

= sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1], color="Red", shade =␣

↪**True**)

= sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0], ax =g, color="Green",␣

g.set\_xlabel("Pregnancies")

g.set\_ylabel("Frequency")

g.legend(["Positive","Negative"])

↪shade= **True**)

**Outcome**

[ ]:

[ ]:



sns.countplot('Outcome',data=df)

There are more people who do not have diabetes in dataset which is around 65% and 35% people has diabetes

**Glucose**



df

[ ]: plt.figure(figsize=(10,6))

sns.violinplot(data=df, x="Outcome", y="Glucose",

split=**True**, inner="quart", linewidth=1)

The chances of diabetes is gradually increasing with level of Glucose

[ ]: *# Explore Glucose vs Outcome*

plt.figure(figsize=(13,6))

= sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade = **True**)

= sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade=␣

↪**True**)

g.set\_xlabel("Glucose")

g.set\_ylabel("Frequency" )



g.legend(["Positive","Negative"])

**Explore Glucose vs BMI vs Age**

## [ ]: # Glucose vs BMI vs Age

plt.figure(figsize=(20,10))

sns.scatterplot(data=df, x="Glucose", y="BMI", hue="Age", size="Age")

As per observation there are some outliers in features. We need to remove outliers in feature engineering

**BloodPressure**

[ ]: *# Explore Age vs Sex, Parch , Pclass and SibSP*

= sns.catplot(y="BloodPressure",x="Outcome",data=df,kind="box") g.set\_ylabels("Blood Pressure")

g.set\_xlabels("Outcome")

**Age vs Outcome**

[ ]: *# Explore Age*

= sns.catplot(y="Age",x="Outcome",data=df,kind="box") g.set\_ylabels("Age")

g.set\_xlabels("Outcome")

**DiabetesPedigreeFunction**

[ ]: sns.set\_theme(style="whitegrid") plt.figure(figsize=(7,5))

sns.boxenplot(x="Outcome",

y="DiabetesPedigreeFunction", color="b", scale="linear", data=df)

g.set\_ylabels("Diabetes Pedigree Function")

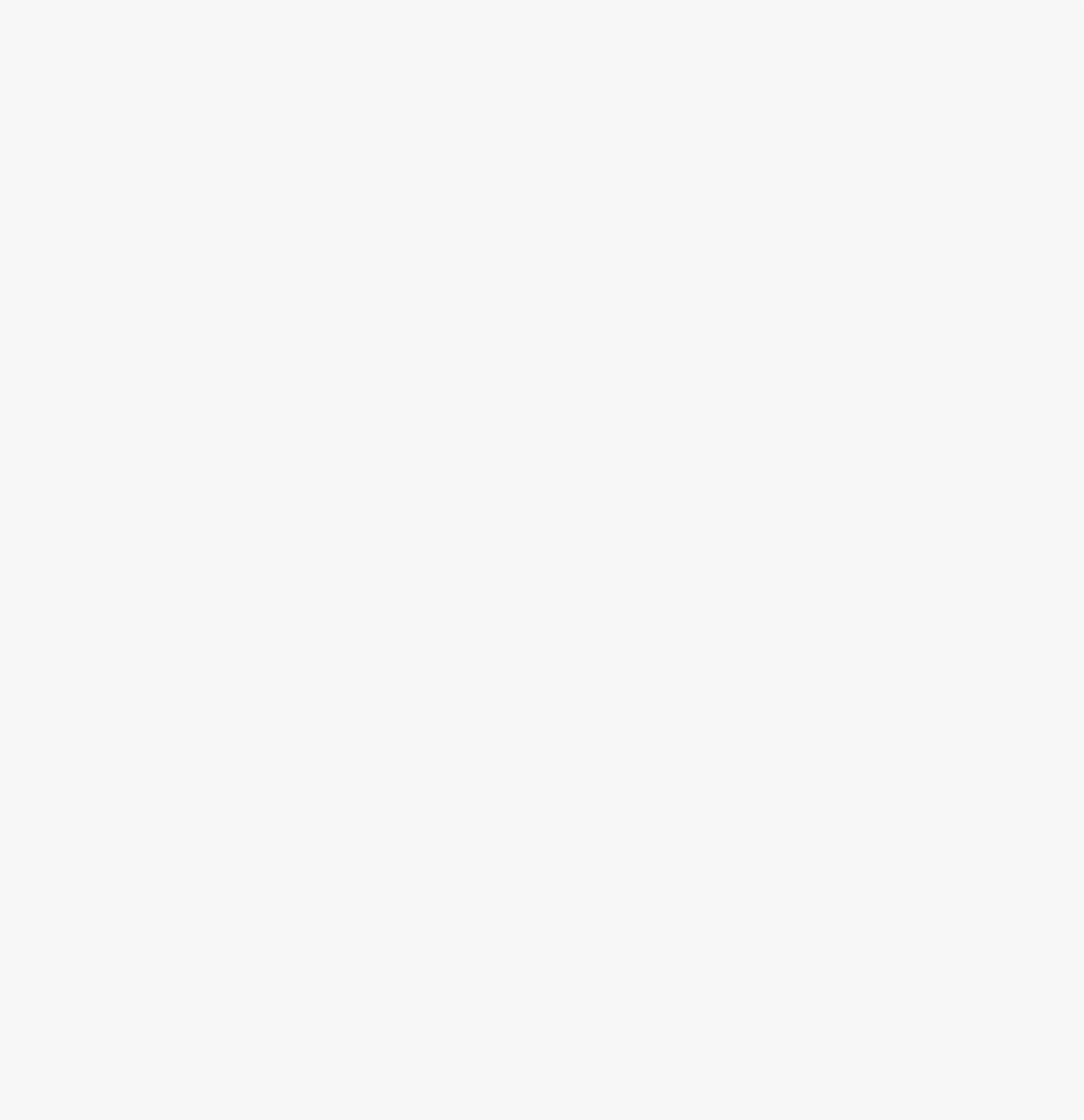
g.set\_xlabels("Outcome")

1. **Feature Enginnering**

Till now, i explored the dataset, did missing value corrections and data visualization. Next, i have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning. Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible.

1. **Outlier Detection**

[ ]: **def** detect\_outliers(df,n,features): outlier\_indices = []



*"""*

*Detect outliers from given list of features. It returns a list of the*␣ ↪*indices*

*according to the observations containing more than n outliers according to the Tukey method*

*"""*

*iterate over features(columns)* **for** col

**in** features:

Q1 = np.percentile(df[col], 25) Q3 = np.percentile(df[col],75)

IQR=Q3-Q1

*# outlier step*

outlier\_step = 1.5 \* IQR

*# Determine a list of indices of outliers for feature col*

outlier\_list\_col = df[(df[col] < Q1 - outlier\_step) | (df[col] > Q3 +␣ ↪outlier\_step )].index

*append the found outlier indices for col to the list of outlier*␣

↪*indices*

outlier\_indices.extend(outlier\_list\_col)

*select observations containing more than 2 outliers*

outlier\_indices = Counter(outlier\_indices)

multiple\_outliers = list( k **for** k, v **in** outlier\_indices.items() **if** v > n )

**return** multiple\_outliers

*# detect outliers from numeric features*

outliers\_to\_drop = detect\_outliers(df, 2 ,["Pregnancies", 'Glucose',␣

↪'BloodPressure', 'BMI', 'DiabetesPedigreeFunction', 'SkinThickness',␣

↪'Insulin', 'Age'])

[ ]: df.loc[outliers\_to\_drop] *# Show the outliers rows*

[ ]: df.drop(df.loc[outliers\_to\_drop].index, inplace=**True**)

I have successfully removed all outliers from dataset now. The next step is to split the dataset in train and test and procceed the modeling

###### Modeling Transforming Data

Before i split the dataset i need to transform the data into quantile using sklearn.preprocessing

[ ]: q = QuantileTransformer()



= q.fit\_transform(df) transformedDF = q.transform(X) transformedDF = pd.DataFrame(X)

transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure',␣ ↪'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age',␣ ↪'Outcome']

[ ]: transformedDF.head()

###### Data Splitting

[ ]: *## Separate train dataset and test dataset* features = df.drop(["Outcome"], axis=1) labels = df["Outcome"]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels,␣ ↪test\_size=0.30, random\_state=7)

1. Cross Validate Models

[ ]: **def** evaluate\_model(models):

*"""*

*Takes a list of models and returns chart of cross validation scores using*␣ ↪*mean accuracy*

*"""*

*Cross validate model with Kfold stratified cross val* kfold = StratifiedKFold(n\_splits = 10)

result = []

**for** model **in** models :

result.append(cross\_val\_score(estimator = model, X = x\_train, y =␣

↪y\_train, scoring = "accuracy", cv = kfold, n\_jobs=4))

cv\_means = [] cv\_std = []

**for** cv\_result **in** result:

cv\_means.append(cv\_result.mean())

cv\_std.append(cv\_result.std())

result\_df = pd.DataFrame({ "CrossValMeans":cv\_means, "CrossValerrors": cv\_std, "Models":[

"LogisticRegression", "DecisionTreeClassifier", "AdaBoostClassifier", "SVC",

"RandomForestClassifier", "GradientBoostingClassifier", "KNeighborsClassifier"

]

})

*# Generate chart*

bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result\_df,␣ ↪orient = "h")

bar.set\_xlabel("Mean Accuracy") bar.set\_title("Cross validation scores") **return** result\_df

[ ]: *# Modeling step Test differents algorithms*

random\_state = 30 models = [

LogisticRegression(random\_state = random\_state, solver='liblinear'), DecisionTreeClassifier(random\_state = random\_state), AdaBoostClassifier(DecisionTreeClassifier(random\_state = random\_state),␣

↪random\_state = random\_state, learning\_rate = 0.2),

SVC(random\_state = random\_state), RandomForestClassifier(random\_state = random\_state), GradientBoostingClassifier(random\_state = random\_state), KNeighborsClassifier(),

]

evaluate\_model(models)

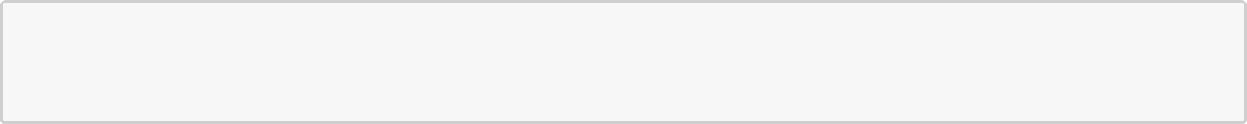
As per above observation, i found that SVC, RandomForestClassifier, and LogisticRegression model has more accuracy. Next, i will do hyper parameter tuning on three models

1. Hyperparameter Tuning

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

I have done tuning process for SVC, RandomForestClassifier, and LogisticRegression models one by one

[ ]:



*# Import libraries*

**from sklearn.model\_selection import** GridSearchCV **from sklearn.metrics import** classification\_report

[ ]:

**def** analyze\_grid\_result(grid\_result):

*'''*

*Analysis of GridCV result and predicting with test dataset Show classification report at last '''*

*# Best parameters and accuracy*

print("Tuned hyperparameters: (best parameters) ", grid\_result.best\_params\_) print("Accuracy :", grid\_result.best\_score\_)

means = grid\_result.cv\_results\_["mean\_test\_score"] stds = grid\_result.cv\_results\_["std\_test\_score"]

**for** mean, std, params **in** zip(means, stds, grid\_result.

↪cv\_results\_["params"]):

print()

print("**%0.3f** (+/-**%0.03f**) for **%r**" % (mean, std \* 2, params))

print("Detailed classification report:") print()

y\_true, y\_pred = y\_test, grid\_result.predict(x\_test) print(classification\_report(y\_true, y\_pred))

print()

First of all i have imported GridSearchCV and classification\_report from sklearn library. Then, i have defined analyze\_grid\_result method which will show prediction result. I called this method for each Model used in SearchCV

1. LogisticRegression

[ ]: *# Define models and parameters for LogisticRegression*

model = LogisticRegression(solver='liblinear')

solvers = ['newton-cg', 'liblinear']

penalty = ['l2']

c\_values = [100, 10, 1.0, 0.1, 0.01]

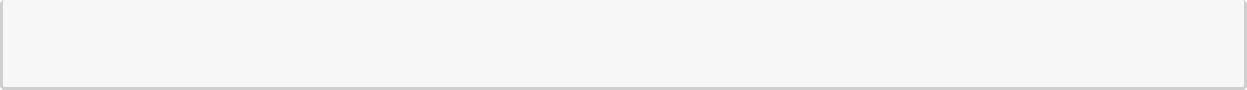
*# Define grid search*

grid = dict(solver = solvers, penalty = penalty, C = c\_values)

cv = StratifiedKFold(n\_splits = 50, random\_state = 1, shuffle = **True**)

grid\_search = GridSearchCV(estimator = model, param\_grid = grid, cv = cv,␣ ↪scoring = 'accuracy', error\_score = 0)

logi\_result = grid\_search.fit(x\_train, y\_train)



*Logistic Regression Hyperparameter Result*

analyze\_grid\_result(logi\_result)

As per my obversation, in LogisticRegression it returned best score 0.78 with {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'} parameters. Next i will perform tuning for other models.

###### SVC

[ ]: *# Define models and parameters for LogisticRegression* model = SVC()

*Define grid search*

tuned\_parameters = [

{"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},

{"kernel": ["linear"], "C": [1, 10, 100, 1000]},

]

cv = StratifiedKFold(n\_splits = 2, random\_state = 1, shuffle = **True**)

grid\_search = GridSearchCV(estimator = model, param\_grid = tuned\_parameters, cv␣ ↪= cv, scoring = 'accuracy', error\_score = 0)

scv\_result = grid\_search.fit(x\_train, y\_train)

*SVC Hyperparameter Result*

analyze\_grid\_result(scv\_result)

SVC Model gave max 0.77 accuracy which is bit less than LogisticRegression. I will not use this model anymore.

1. RandomForestClassifier

[ ]: *# Define models and parameters for LogisticRegression*

model = RandomForestClassifier(random\_state=42)

*# Define grid search*

tuned\_parameters = {

'n\_estimators': [200, 500],

'max\_features': ['auto', 'sqrt', 'log2'],

'max\_depth' : [4,5,6,7,8],

'criterion' :['gini', 'entropy']

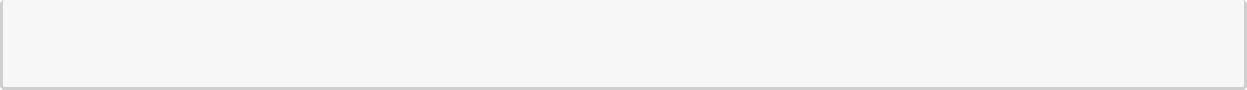
}

cv = StratifiedKFold(n\_splits = 2, random\_state = 1, shuffle = **True**)

grid\_search = GridSearchCV(estimator = model, param\_grid = tuned\_parameters, cv␣

↪= cv, scoring = 'accuracy', error\_score = 0)

grid\_result = grid\_search.fit(x\_train, y\_train)



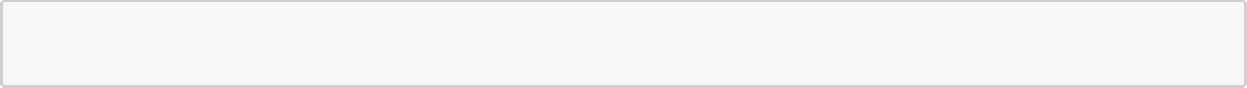
*SVC Hyperparameter Result*

analyze\_grid\_result(grid\_result)

Randomforest model gave max 0.76% accuracy which is not best comparing to other model. So i decided to use LogisticRegression Model for prediction

###### Prediction

Till now, i worked on EDA, Feature Engineering, Cross Validation of Models, and Hyperparameter Tuning and find the best working Model for my dataset. Next, I did prediction from my test dataset and storing the result in CSV

[ ]: y\_pred = logi\_result.predict(x\_test) print(classification\_report(y\_test, y\_pred))

[ ]: x\_test['pred'] = y\_pred x\_test

**Analysis:**

1. Calculate the BMI (Body Mass Index) from the 'BMI' and 'SkinThickness' columns in the diabetes dataset:

import pandas as pd

df = pd.read\_csv("diabetes.csv")

# Calculate BMI def

calculate\_bmi(row):

if

row['SkinThickness'] == 0:

return 0

else:

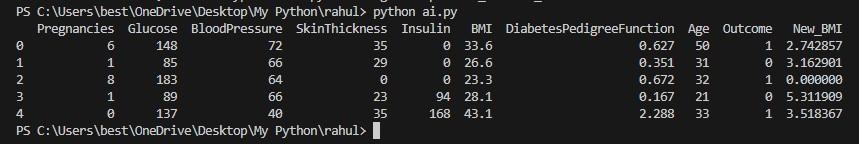
bmi = (row['BMI'] \* 100) / (row['SkinThickness'] \*\* 2) return bmi

df['New\_BMI'] = df.apply(calculate\_bmi,

axis=1) df.to\_csv("diabetes\_with\_bmi.csv", index=False) print(df.head())

**Output:**

(new bmi data in github code)



1. Model Hyperparameter Tuning.

import pandas as pd from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.preprocessing import StandardScaler from sklearn import svm

df = pd.read\_csv("diabetes.csv")

X = df.drop("Outcome", axis=1) y

= df["Outcome"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=769)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test)

svm\_classifier = svm.SVC() param\_grid = {

'C': [0.1, 1, 10],

'kernel': ['linear', 'rbf'],

'gamma': [0.1, 1, 10]

} grid\_search = GridSearchCV(svm\_classifier, param\_grid, cv=5, n\_jobs=-1); grid\_search.fit(X\_train, y\_train); best\_params = grid\_search.best\_params\_; best\_svm\_model = svm.SVC(C=best\_params['C'], kernel=best\_params['kernel'],gamma=best\_params['gamma']); best\_svm\_model.fit(X\_train, y\_train) accuracy = best\_svm\_model.score(X\_test, y\_test)

print("Best Model Accuracy:", accuracy) print("Best Hyperparameters:", best\_params)

#### Output:

1. Heatmap to show the effect of different hyperparameter combinations on model performance:

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.preprocessing import StandardScaler from sklearn import svm

# Load the diabetes dataset

df = pd.read\_csv("diabetes.csv")

X = df.drop("Outcome", axis=1) y

= df["Outcome"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=769)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test) svm\_classifier = svm.SVC()

# Define a parameter grid for Grid Search param\_grid = {

'C': [0.1, 1, 10],

'kernel': ['linear', 'rbf'],

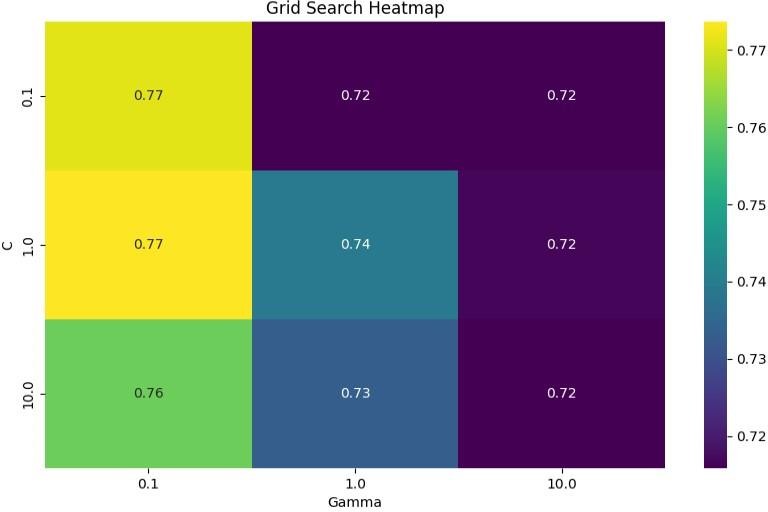
'gamma': [0.1, 1, 10]

} grid\_search = GridSearchCV(svm\_classifier, param\_grid, cv=5, n\_jobs=-1) grid\_search.fit(X\_train, y\_train) best\_params = grid\_search.best\_params\_ best\_svm\_model = svm.SVC(C=best\_params['C'], kernel=best\_params['kernel'], gamma=best\_params['gamma'])

best\_svm\_model.fit(X\_train, y\_train)

# Create a heatmap to visualize the results of hyperparameter tuning results = pd.DataFrame(grid\_search.cv\_results\_) results\_pivot = results.pivot\_table(index='param\_C', columns='param\_gamma', values='mean\_test\_score') plt.figure(figsize=(10, 6)) sns.heatmap(results\_pivot, annot=True, cmap='viridis') plt.title("Grid Search Heatmap") plt.xlabel("Gamma") plt.ylabel("C") plt.show()

#### Output:



1. 'Glucose' and 'Age' Distribution:

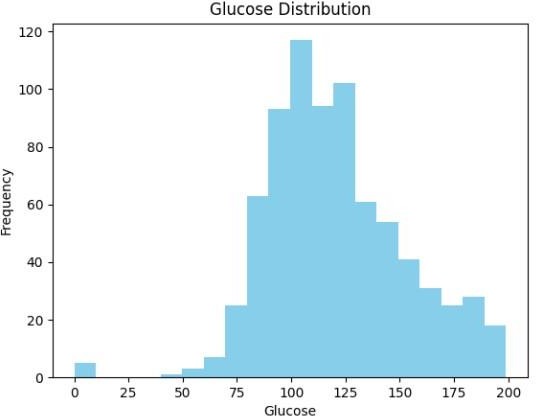
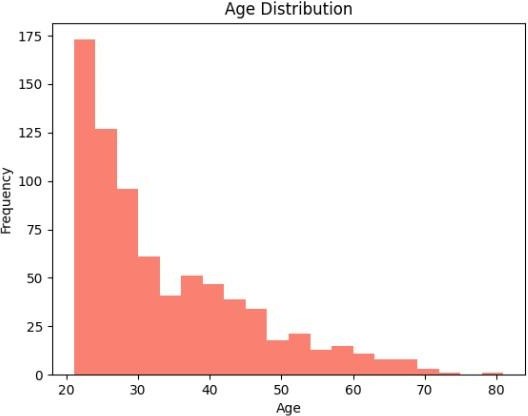
import pandas as pd import matplotlib.pyplot as plt

df = pd.read\_csv("diabetes.csv")

# Create a histogram for the 'Glucose' column plt.hist(df['Glucose'], bins=20, color='skyblue') plt.title('Glucose Distribution') plt.xlabel('Glucose') plt.ylabel('Frequency') plt.show()

# Create a histogram for the 'Age' column plt.hist(df['Age'], bins=20, color='salmon') plt.title('Age Distribution') plt.xlabel('Age') plt.ylabel('Frequency') plt.show()

#### Output:



1. Correlation Heatmap useful for identifying which features are most correlated with the target variable 'Outcome':

import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

# Load the diabetes dataset df

= pd.read\_csv("diabetes.csv")

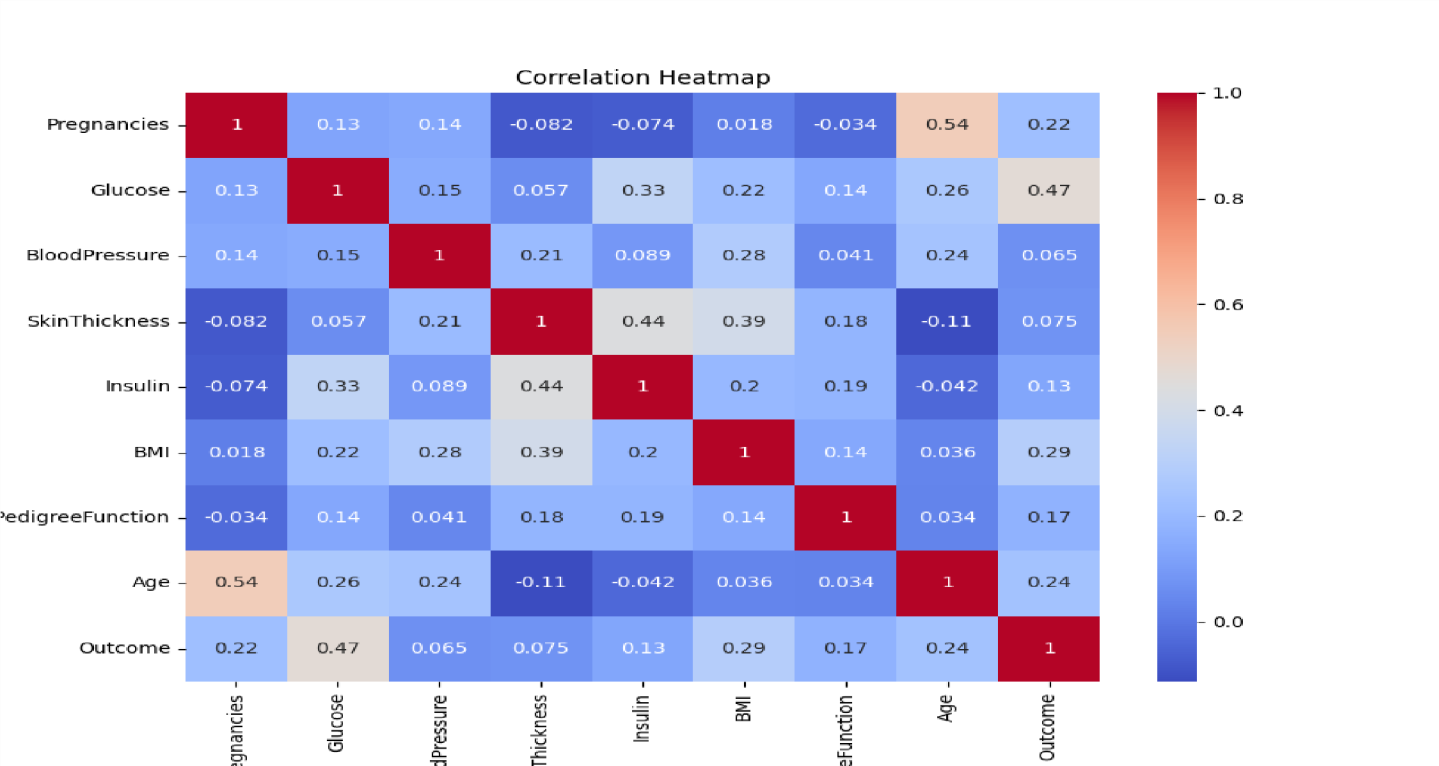
# Calculate the correlation matrix correlation\_matrix

= df.corr()

# Create a heatmap to visualize the correlations plt.figure(figsize=(10, 8)) sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm') plt.title('Correlation Heatmap') plt.show()

#### Output:

##### Data Cleaning:



We started by cleaning the dataset. Missing values were filled with the mean value of their respective columns, and duplicate rows were removed, ensuring the data is ready for analysis.

##### Data Analysis:

We conducted some basic data analysis:

* + - Summary statistics of the dataset to understand the distribution of numerical features.
    - Examined the class distribution to understand the balance between positive (diabetic) and negative (nondiabetic) cases.

##### Modeling:

We used a Support Vector Machine (SVM) classifier with a linear kernel to build a diabetes prediction model. The model achieved an accuracy score on the test data.

##### Hyperparameter Tuning:

. We optimized the hyperparameters of the SVM model using Grid Search to find the best combination of hyperparameters for improved model performance.

##### Feature Selection:

We performed feature selection using Recursive Feature Elimination (RFE) with a Random Forest classifier to identify the most important features for the model.

##### Data Visualization:

We created visualizations, including histograms to show the distribution of 'Glucose' and 'Age' features and a correlation heatmap to visualize feature relationships.

# Conclusion:

Prediction System aims to provide a valuable tool for healthcare professionals and individuals to assess diabetes risk early, enabling timely interventions and improved diabetes Management.

After using all these patient records, we are able to build a machine learning model (random forest – best one) to accurately predict whether or not the patients in the dataset have diabetes or not along with that we were able to draw some insights from the data via data analysis and visualization.