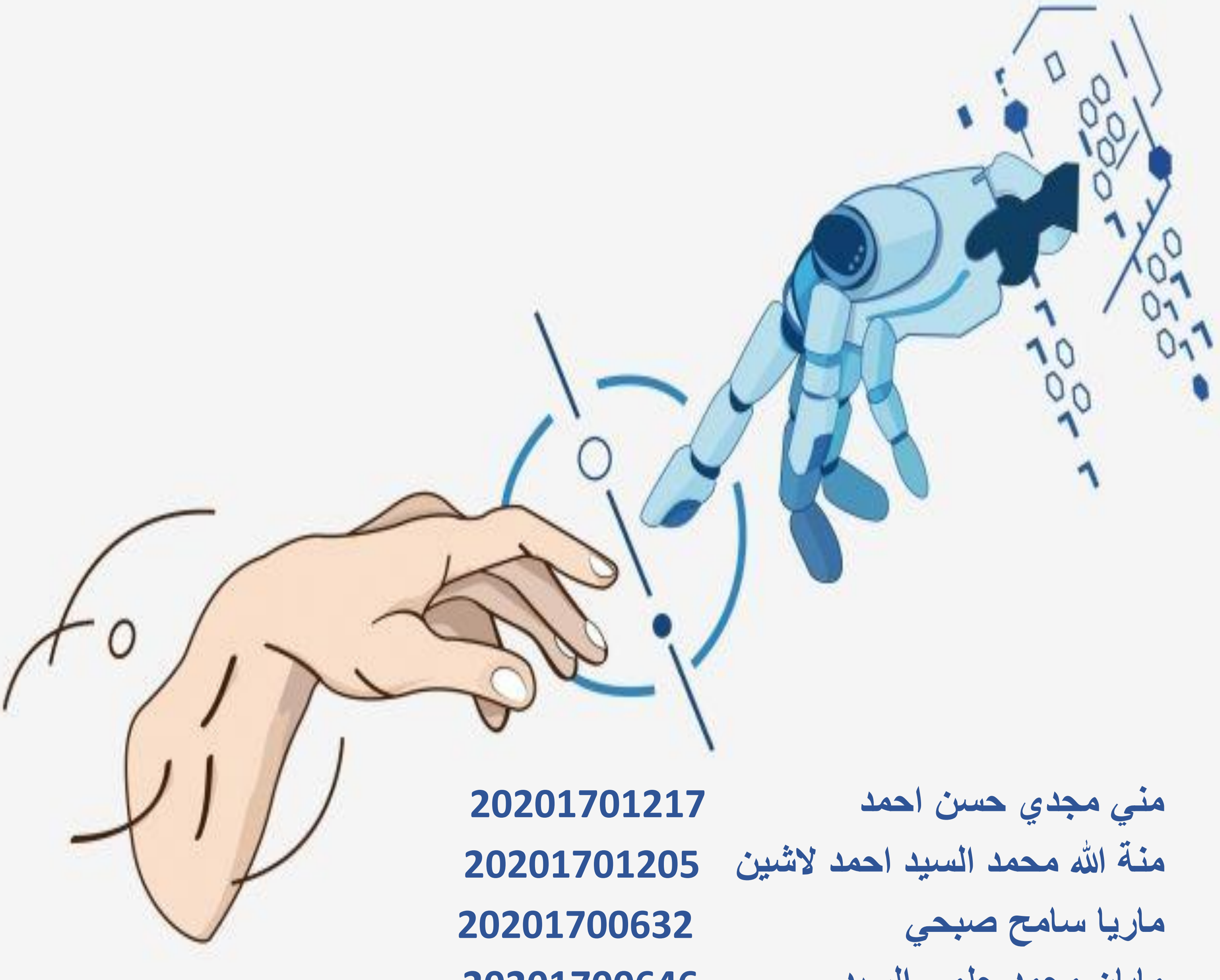
Diabetes Health Indicators



مني مجدي حسن احمد	20201701217
منة الله محمد السيد احمد لاشين	20201701205
ماريا سامح صبحي	20201700632
مايان محمد حلمي السيد	20201700646
مارلين ايهاب رزق جرجس	20201700630
عبد الرحمن طارق سيد	20201700441
مارينا نادي شحاتة عوض	20201700636

Project Brief:

-Diabetes mellitus is one of the most serious illnesses in the world.
-Project goal: predict whether a person has diabetes or not, if he does, then the output (Diabetes _binary) will be 1 and if not, then the output will be 0.

Preprocessing:

Data collection:

Data is read from: **diabetes _ binary _ health _ indicators _ BRFSS2015.csv**

Data cleansing:

drop_duplicates(): function is used to remove any duplicates rows.

isnull().sum(): to check if the data contains null values or not.

```
null values are
 Diabetes_binary         0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
```

Feature extraction:

-Features are selected by calculating their correlation then selecting features with correlation greater than 0.17, then it's visualized using heatmap.

Data Scaling:

-It is a technique used to **standardize** the independent features present in the data **within a fixed range**. It is performed during the data pre-processing to handle highly varying values.

- **Method Used**:

Standardization: It is a technique that re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1.

```
# Datascaling by standard scaler
from sklearn.preprocessing import StandardScaler
fig, ax = plt.subplots(figsize=(12, 4))
scaler=StandardScaler()
ssx=scaler.fit_transform(x)
ax.scatter(ssx[:,0],y)
ax.scatter(ssx[:,1],y)
plt.title( "scatter plot of standard scaler")
plt.show()
```

Balancing Dataset:

- Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations i.e., one class label has a very high number of observations and the other has a very low number of observations.

- Oversampling can be performed by increasing the amount of minority class instances or samples with producing new instances or repeating some instances.

```
# Balancing
from collections import Counter
print("...............................................................................")
print(f"Training target statistics before oversampling: {Counter(y_train)}")
from imblearn.over_sampling import RandomOverSampler
over_sampler = RandomOverSampler(random_state=42)
X_res, y_res = over_sampler.fit_resample(x_train, y_train)
print(f"Training target statistics after oversampling: {Counter(y_res)}")
```

Imported libraries:

1)matplotlib: To draw the figure and figure function is used to determine figure size
2) seaborn: To draw heatmap
3)joblib: To load and save the model

Model Training:

Logistic regression:

- ✓ The **logistic regression** model is used to solve classification problems, and deals with discrete values and data.
- ✓ For the small data**: liblinear** solver (identify the size of dataset) is used, while for the big data: **sag/saga** solver is used.
- ✓ Most important functions:
  - **Fit()**: Pass training data as parameters.
  - **Predict()**
  - **Score():** Calculate accuracy.
- ✓ **Accuracy:** 86.53%
- ✓ **Precision, recall, f1_score:**

	precision	recall	f1-score
0.0	0.8787	0.9789	0.9261
1.0	0.5409	0.1551	0.2410

```
# # training and fitting Logistic Regression to model
 from sklearn.linear_model import LogisticRegression

 lr = LogisticRegression(solver="saga", random_state=0)
 lr.fit(x_train, y_train)

# # Saving model into a file
 joblib.dump(lr,'logistic_regression_model')

# Loading the model
lr_loaded = joblib.load("logistic_regression_model")
# Making predicition on logisitic regression
lr_pre = lr_loaded.predict(x_test)
# Calculating accuracy of logisitic regression
accuracy = lr_loaded.score(x_test, y_test)
print("accuracy of lr= ", accuracy * 100, "%")
# Making confusion matrix on logisitic regression
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, lr_pre)
print(cm)
h = sns.heatmap(cm, annot=True, cmap="RdYlGn", fmt="d")
plt.xlabel("Predicted values")
plt.ylabel("Actual values")
plt.show()
```

Support vector machine (SVM):

- ✓ **SVM** is a binary classification model whose basic model is a linear classifier defined in the eigenspace with the largest interval.
- ✓ **Linear SVC** (Straight lines are used to divide data into different classes.) is used in this module.
- ✓ **Accuracy:** 86.39%
- ✓ **Precision, recall, f1_score:**

	precision	recall	f1-score
0.0	0.8684	0.9925	0.9263
1.0	0.5618	0.0597	0.1080

```
# # training and fitting SVM to model
 from sklearn.svm import LinearSVC

 sv = LinearSVC()
sv.fit(x_train, y_train)

# # Saving model into a file
 joblib.dump(sv,'svm_model')

# Loading the model
sv_loaded = joblib.load("svm_model")
# Making prediction of svm
svm_pre = sv_loaded.predict(x_test)
print(svm_pre.shape)
# calculating accuracy of svm
accuracy = sv_loaded.score(x_test, y_test)
print("accuracy of svm= ", accuracy * 100, "%")
# Making confusion matrix on svm
cmofsvm = confusion_matrix(y_test, svm_pre)
print(cmofsvm)
h = sns.heatmap(cmofsvm, annot=True, cmap="RdYlGn", fmt="d")
plt.xlabel("Predicted values")
plt.ylabel("Actual values")
plt.show()
```

Decision tree:

ID3 Algorithm: Using this method, decision trees are built iteratively by finding out the maximum Information Gain among all the featured data columns to be represented as the node of the tree.

-Information gain is considered as Entropy "Errors" reduction.

-As Information gain increases, Entropy decreases.

- ✓ **Accuracy:** 84.3%
- ✓ **Precision, recall, f1_score:**

	precision	recall	f1-score
0.0	0.8819	0.9440	0.9119
1.0	0.3746	0.2097	0.2689

```
# # training and fitting Decision tree to model
 from sklearn.tree import DecisionTreeClassifier

 dt = DecisionTreeClassifier()
 print(dt.fit(x_train, y_train))

# # Saving model into a file
 joblib.dump(dt,'decision_tree_model')

# Loading model
dt_loaded = joblib.load("decision_tree_model")

# Making predictions on decision tree
dt_pre = dt_loaded.predict(x_test)
print(dt_pre.shape)
# calculating accuracy of decision tree
accuracy = dt_loaded.score(x_test, y_test)
print("accuracy of dt = ", accuracy * 100, "%")
# Making confusion matrix on decision tree
cmofdt = confusion_matrix(y_test, dt_pre)
print(cmofdt)
h = sns.heatmap(cmofdt, annot=True, cmap="RdYlGn", fmt="d")
plt.xlabel("Predicted values")
plt.ylabel("Actual values")
plt.show()
```