APPENDIX

CODING IMPORT LIBRARIES

```
import pandas as pd
import numpy as np

import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder, StandardScaler,
normalize
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

CODING IMPORT DATA & DATA PREPROCCESSING

```
file = pd.read_csv(r'Data/diabetes.csv')
file.info()
plt.figure(figsize=(20,20))
sns.heatmap(file[:-1].corr(), annot=True,cbar=False)
plt.show()
sns.distplot(file['Pregnancies'])
sns.distplot(file['Glucose'])
sns.distplot(file['BloodPressure'])
sns.distplot(file['SkinThickness'])
sns.distplot(file['Insulin'])
sns.distplot(file['BMI'])
sns.distplot(file['DiabetesPedigreeFunction'])
sns.distplot(file['Age'])
```

CODING MENUNJUKAN INFORMASI DATA

```
def check_file(file):
    print("###################### Shape ######################")
    print(file.shape)
    print("###################### Types ######################")
    print(file.dtypes)
    print("###################### Head ######################")
    print(file.head(3))
    print("###################### Tail ######################")
    print(file.tail(3))
    print("###################### NA ######################")
    print(file.isnull().sum())
    print("###################### Quantiles ######################")
    print(file.quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T)
```

```
check_file(file)
```

CODING FEATURE ENGINEERING

```
file.loc[file['BMI'] == 0]
file.loc[file['Glucose'] == 0]
file.loc[file['BloodPressure'] == 0]
file.loc[file['SkinThickness'] == 0]
drop_index = file.loc[(file['BMI'] == 0) & (file['BloodPressure']
  == 0) & (file['SkinThickness'] == 0) & (file['Insulin'] ==
  0), :].index
file.drop(drop_index, axis = 0, inplace = True)
for i in file.columns.tolist():
    print(i, '-', len(file.loc[file[i] == 0, :]))
file.sample(20)
file['BMI'] = np.where(file['BMI'] == 0, np.nan, file['BMI'])
file['Glucose'] = np.where(file['Glucose'] == 0, np.nan,
  file['Glucose'])
file['BloodPressure'] = np.where(file['BloodPressure'] == 0,
  np.nan, file['BloodPressure'])
file['SkinThickness'] = np.where(file['SkinThickness'] == 0,
  np.nan, file['SkinThickness'])
file['BMI'].fillna(27, inplace = True)
file['Glucose'].fillna(file['Glucose'].mean(), inplace = True)
file['BloodPressure'].fillna(file['BloodPressure'].mean(), inplace
  = True)
file['SkinThickness'].fillna(file['SkinThickness'].mean(), inplace
  = True)
file.describe()
file.loc[file['SkinThickness'] > 60]
file.drop(file.loc[file['SkinThickness'] > 60].index, axis = 0,
  inplace = True)
file.loc[file['BMI'] > 55]
file.drop(file.loc[file['BMI'] > 55].index, axis = 0, inplace =
  True)
```

CODING SCALING DATA

```
file_scaled = scaler.fit_transform(file.iloc[:,:-1])
file_scaled.shape
file.shape
```

CODING MEMBAGI DATA DAN TRAINING 10% DARI DATA

```
from sklearn.model_selection import train_test_split
def train_valid(data, frac= 0.1):
  data_x = data
  data_y = file['Outcome']

  train_x, valid_x, train_y, valid_y
  train_test_split(data_x,data_y,test_size=frac)
  return train_x,valid_x,train_y,valid_y
```

```
train_x, valid_x, train_y, valid_y = train_valid(file_scaled)scaler
  = StandardScaler()
train_x = np.asarray(train_x)
train_y = np.asarray(train_y)
valid_x = np.asarray(valid_x)
train_y
```

CODING MELIHAT OUTCOME DARI TRAINING

```
import seaborn as sns
sns.countplot(file['Outcome']))
```

CODING MENENTUKAN ROC CURVE

```
import matplotlib.pyplot as plt
def plot_roc_cur(fper, tper):
      plt.plot(fper, tper, color='orange', label='ROC')
      plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('Receiver Operating Characteristic (ROC) Curve')
      plt.legend()
      plt.show()
```

CODING NAÏVE BAYES MEMAKAI SKLEARN

```
import time
from   sklearn.metrics   import   accuracy_score,   roc_auc_score,
plot_confusion_matrix, roc_curve, classification_report
def   run_model(model,   X_train,   y_train,   X_test,   y_test,
verbose=True):
      t0=time.time()
      if verbose == False:
            model.fit(X_train,y_train, verbose=0)
      else:
               model.fit(X_train,y_train)
      y_pred = model.predict(X_test)
      accuracy = accuracy_score(y_test, y_pred)
      roc_auc = roc_auc_score(y_test, y_pred)
      time_taken = time.time()-t0
      print("Accuracy = {}".format(accuracy))
      print("ROC Area under Curve = {}".format(roc_auc))
      print("Time taken = {}".format(time_taken))
      print(classification_report(y_test,y_pred,digits=5))

      probs = model.predict_proba(X_test)
      probs = probs[:, 1]
      fper, tper, thresholds = roc_curve(y_test, probs)
      plot_roc_cur(fper, tper)

      plot_confusion_matrix(model,X_test,y_test,cmap=plt.cm.Blues,
      normalize = 'all')
```

```
    return model, accuracy

from sklearn.naive_bayes import GaussianNB

NB =GaussianNB()

NB, accuracy_NB = run_model(NB,train_x,train_y,valid_x,valid_y)
```

CODING LOGISTIC REGRESSION TANPA LIBRARY

```
class LogisticRegression:
      def __init__(self,lr=0.01,epochs=30):
            self.lr = lr
            self.epochs = epochs
            self.weights = None

      def sigmoid(self,z):
            return 1/(1+ np.e**(-z))

      def cost_function(self,X,y):
            z = np.dot(X,self.weights)
            predict1 = y*np.log(self.sigmoid(z))
            predict0 = (1 - y) * np.log(1 - self.sigmoid(z))
            return -sum(predict1 + predict0) / len(X)

      def fit(self,X,y):
            loss = []
            self.weights = np.random.rand(X.shape[1])
            n =len(X)

            for i in range(self.epochs):
                yhat = self.sigmoid(np.dot(X,self.weights))
                self.weights -= self.lr*np.dot(X.T,yhat-y)/n
                loss.append(self.cost_function(X,y))

            self.weights = self.weights
            self.loss = loss

      def predict(self,X):
            z = np.dot(X, self.weights)

            return [1 if i > 0.5 else 0 for i in self.sigmoid(z)]

def model_accuracy(preds,y_true):
      ret = (preds == y_true)
      return sum(ret)/len(preds)
model = LogisticRegression()
model.fit(train_x, train_y)
preds = model.predictC)
```

CODING MELIHAT PREDICTION

```
preds
```

CODING MELIHAT LOSS DARI MODEL

```
xs = np.arange(len(model.loss))
ys = model.loss

plt.plot(xs, ys, lw=3, c='#087E8B')
plt.title('Loss per iteration (MSE)', size=20)
plt.xlabel('Iteration', size=14)
plt.ylabel('Loss', size=14)
plt.show()
```

CODING MELIHAT LOSS PADA MASING-MASING LEARNING RATE

```
losses = {}
for lr in [0.5, 0.1, 0.01, 0.001]:
   model = LogisticRegression(lr = lr)
   model.fit(train_x, train_y)
   losses[f'LR={str(lr)}'] = model.loss

xs = np.arange(len(model.loss))

plt.plot(xs, losses['LR=0.5'], lw=3, label=f"LR = 0.5, Final =
{losses['LR=0.5'][-1]:.2f}")
plt.plot(xs, losses['LR=0.1'], lw=3, label=f"LR = 0.1, Final =
{losses['LR=0.1'][-1]:.2f}")
plt.plot(xs, losses['LR=0.01'], lw=3, label=f"LR = 0.01, Final =
{losses['LR=0.01'][-1]:.2f}")
plt.plot(xs, losses['LR=0.001'], lw=3, label=f"LR = 0.001, Final =
{losses['LR=0.001'][-1]:.2f}")
plt.title('Loss per iteration (MSE) for different learning rates',
size=20)
plt.xlabel('Iteration', size=14)
plt.ylabel('Loss', size=14)
plt.legend()
plt.show()
```

CODING DEFINISI DARI RUN MODEL LOGISTIC REGRESSION

```
def run_model_lr(model, X_train, y_train, X_test, y_test,
verbose=True):
          t0=time.time()
          if verbose == False:
              model.fit(X_train,y_train, verbose=0)
          else:
              model.fit(X_train,y_train)
          y_pred = model.predict(X_test)
          accuracy = accuracy_score(y_test, y_pred)
          roc_auc = roc_auc_score(y_test, y_pred)
          time_taken = time.time()-t0
          print("Accuracy = {}".format(accuracy))
```

```
        print("ROC Area under Curve = {}".format(roc_auc))
        print("Time taken = {}".format(time_taken))
        print(classification_report(y_test,y_pred,digits=5))
    return model, accuracy
```

CODING MENUNJUKAN ACCURACY PADA LOGISTIC REGRESSION

```
model = LogisticRegression(lr=0.54)
model,accuracy_model=
run_model_lr(model,train_x,train_y,valid_x,valid_y)
```

PLAGIARISM CHECK.ORG

6.41% PLAGIARISM APPROXIMATELY

Report #13353757

1 2 3 4 17 Introduction Background Diabetes is a chronic disease characterized by high blood sugar (glucose) levels that occurs as the result of the pancreas not producing enough insulin or the body cannot effectively use the insulin. 1 2 3 4 12 According to WHO international in 2014, 8.5% of adults aged 18 years and older had diabetes. 1 2 3 4 12 23 In 2016, diabetes was the direct cause of 1.6 million deaths and in 2012 high blood glucose was the cause of another 2.2 million deaths. It is a very serious problem that needs to be solved, the cure has not been found yet, but we still could prevent it by diagnosing people who have diabetes disease. If people positively have a diabetes disease, they can quickly prevent it from a young age by watching their diet. So, it is an important thing to predict the patient who has diabetes disease. Nowadays, this problem could be solved using machine learning by training the dataset of patients with and without the symptoms of diabetes. In this case, the dataset will be trained with two different algorithms to predict the onset