Saurabh Rajeshkumar Panchal
Student ID: 015219971

CMPE 257 Lab 1

Task 1

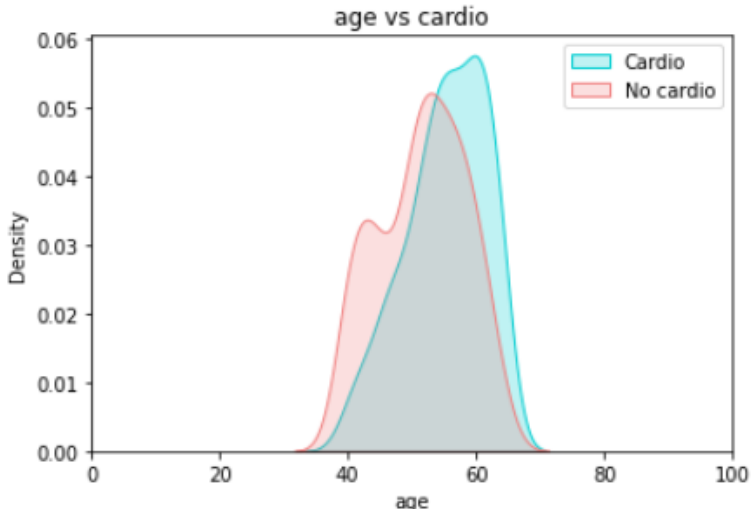
- Task 1 comprised of performing some EDA (Exploratory Data Analysis) on the given cardiovascular data.
- There are 13 columns in our dataset. The Kaggle page describes the meaning of each column like this:
  id -----> id
  age -----> Age(days)
  gender -----> Gender
  height -----> Height(cm)
  weight -----> Weight(kg)
  ap_hi -----> Systolic blood pressure
  ap_lo -----> Diastolic blood pressure
  cholesterol -----> Cholesterol
  gluc -----> Glucose
  smoke -----> Smoking
  alco -----> Alcohol intake
  active -----> Physical activity
  cardio -----> Presence(1) or absence(0) of cardiovascular disease (Target Variable)

- The given features can be divided into 3 categories:
  1. Nominal (Categorical) features : gender, cholesterol, gluc, smoke, alco, active
  2. Numeric (Continuous) features : age, height, weight, ap_hi, ap_lo
  3. Target variable : cardio
- Null values: We can see presence of NaN(missing) values in all columns except 'id' and 'cardio'. Most NaN values are in 'height' column as non-null count is very low.
- Data types: Float values are present in 'age', 'height', 'weight', 'ap_hi', 'ap_lo', 'smoke', 'alco', and 'active'. These columns are numeric and continuous. Int values are present in 'id' and 'cardio' columns. For columns 'gender', 'cholesterol' and 'gluc' we see data type as object because these are categorical features with string values.

- Age analysis:
  age is given in days, we can convert it to years by age = age/365
  As per results of Q4:
  a. Oldest person in the data is : 64.0 years
  b. Youngest person in the data is : 39.0 years
  c. Average age of a person in the data is : 53.4 years

d. Median age of a person in the data is : 54.0 years

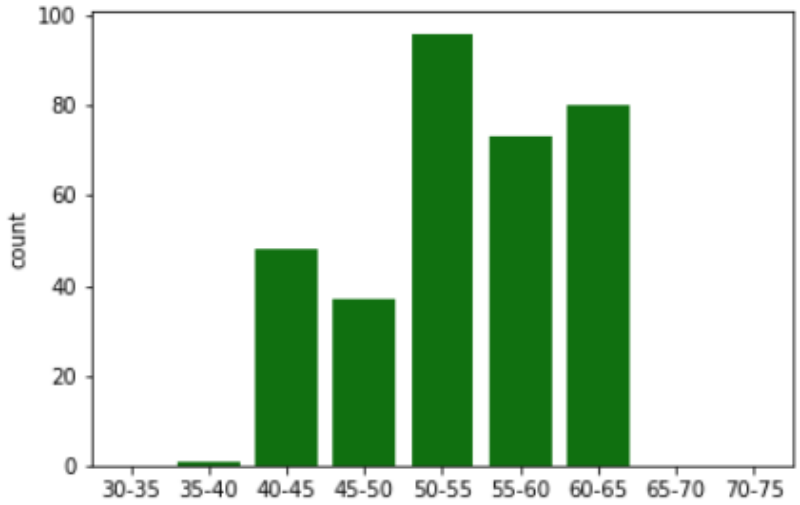
Relation of age vs cardio:



As we can see from the density plot, age density distribution of people with cardio disease is on the right side of density distribution of people with no cardio disease. This suggests that people with higher age(>55) have more risk for cardio disease than people with lesser age(<45).

Dividing records into age groups of "30-35", "35-40", "40-45", "45-50", "50-55", "55-60", "60-65", "65-70", "70-75":
Q4f gives an idea of how many people survived in each age group.
creating bins of 335 people (with non-null age)
In age group 30-35 , total people: 0 , people with no cardio: 0 , survival rate: 0
In age group 35-40 , total people: 1 , people with no cardio: 1 , survival rate: 1.0
In age group 40-45 , total people: 48 , people with no cardio: 36 , survival rate: 0.75
In age group 45-50 , total people: 37 , people with no cardio: 18 , survival rate: 0.49
In age group 50-55 , total people: 96 , people with no cardio: 51 , survival rate: 0.53
In age group 55-60 , total people: 73 , people with no cardio: 34 , survival rate: 0.47
In age group 60-65 , total people: 80 , people with no cardio: 22 , survival rate: 0.28
In age group 65-70 , total people: 0 , people with no cardio: 0 , survival rate: 0
In age group 70-75 , total people: 0 , people with no cardio: 0 , survival rate: 0

Distribution of people in each age group:
People can be divided into bins from age 35-40 to 60-65 as we have people from age 39 to 64 years in our data. We have highest count of people from age group 50-55. 75% of people from age group 40-45 (36 out of 38) have cardio=0, which makes it age group with highest survival rate.



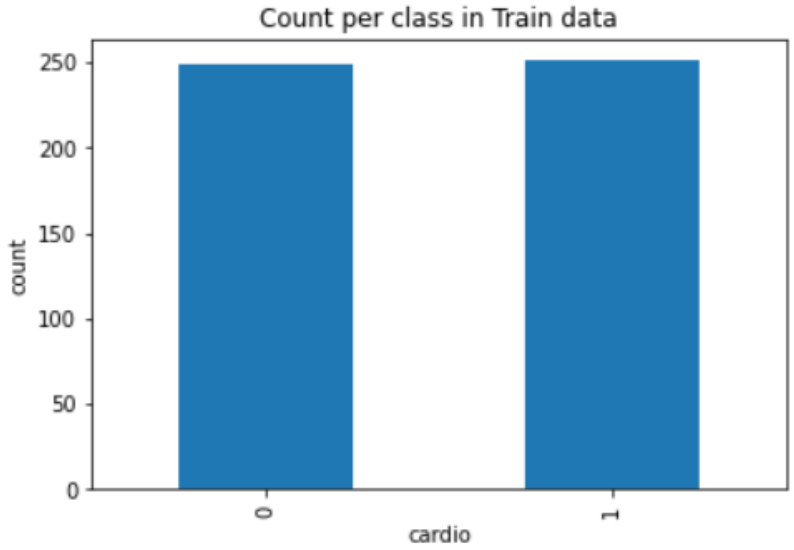
- Processing categorical columns:
  Cholesterol, gluc and gender are categorical columns in our data. We can user LabelEncoder to replace their string values by integer.
  cholesterol unique values: [nan 'Normal' 'High' 'Above Normal']
  glucose unique values: ['Normal' nan 'High' 'Above Normal']
  gender unique values: [nan 'Men' 'Women']

  Example of LabelEncoder:

```
from sklearn.preprocessing import LabelEncoder
cardio_data_train["cholesterol"] = LabelEncoder().fit_transform(cardio_data_train["cholesterol"
cardio_data_train["gluc"] = LabelEncoder().fit_transform(cardio_data_train["gluc"])
cardio_data_train["gender"] = LabelEncoder().fit_transform(cardio_data_train["gender"])
```

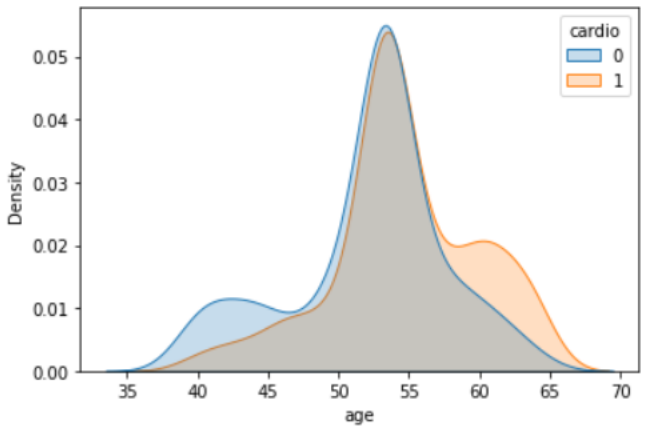
  Alternatively, we can use pandas map lambda function to replace each “Men” with 0, “Women” with 1 and in similar manner replace “Normal” with 0, “Above Normal” with 1, and “High” with 2.
- Graph of distribution of records with cardio = 0 and cardio = 1:

```
Train data:
Number of data points for class_0: 249
Number of data points for class_1: 251
```



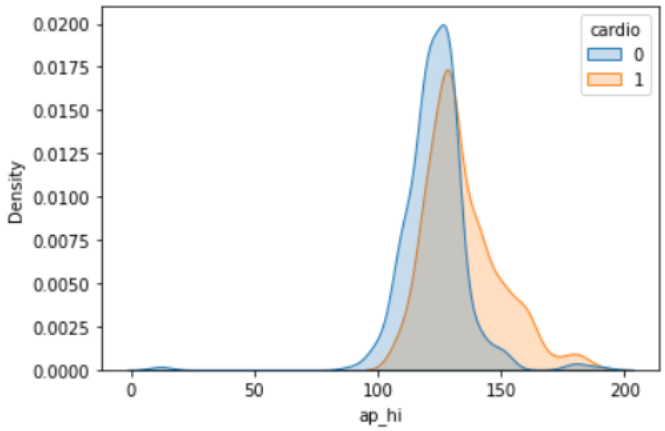
- "id" column is not needed for prediction task since its just identifier for each person and is unique for every record. We can drop this column before feeding training data to classifier.
- For further questions (density plots), we replace numeric columns with mean of that column and replace categorical columns with mode (most occurring value) of the column. This way we can visualize distribution on entire training data.
- In question 4g, we plot relationships between a few features against cardio. The graphs and observations are as mentioned below:

  Age vs Cardio:



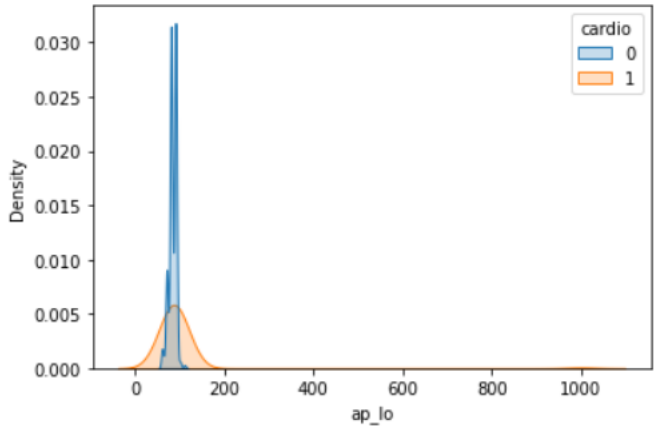
  We can see in the kde plot that people arround the age 35-45 have higher chance of cardio=0. People around the age 50-55 have high survival rate as well as high fatality rate. People with age 57-70 have high chance of having cardio=1.

Ap_hi vs cardio:



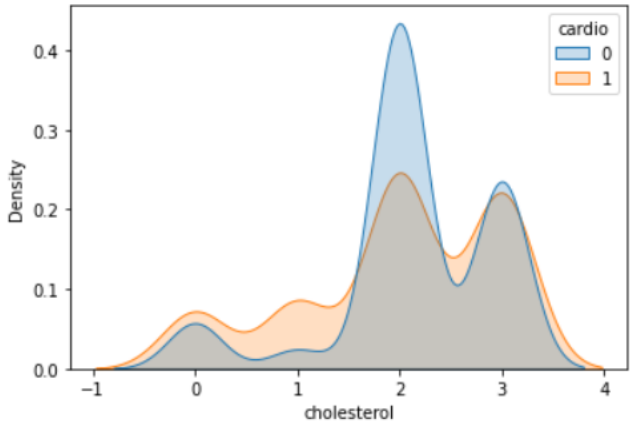
People with average Systolic blood pressure(ap_hi) have higher chance of having cardio=0, i.e. there is higher chance that they don't have cardio disease.

Ap_lo vs cardio:



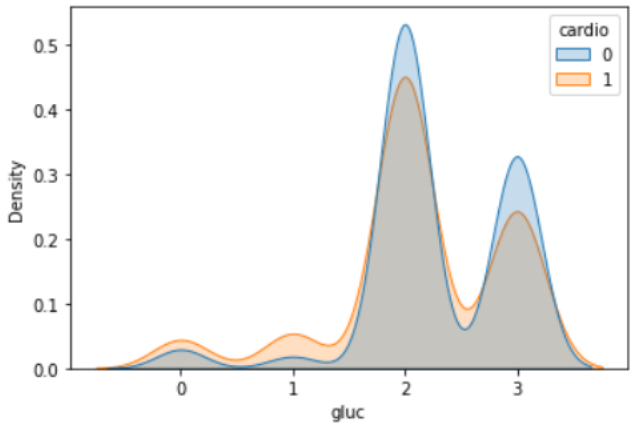
We can see from density plot of ap_lo that people with average Diastolic blood pressure(ap_lo) have very high probability of not having cardio disease.

Cholesterol vs cardio:



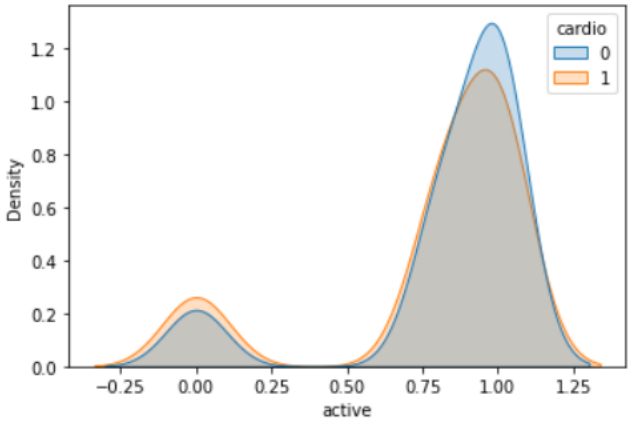
From density plot of cholesterol, we can see that people with Normal(0) cholesterol have higher chance of cardio=0. i.e. there is high chance that they will not have cardio disease, than people who have more cholesterol.

Gluc vs cardio:



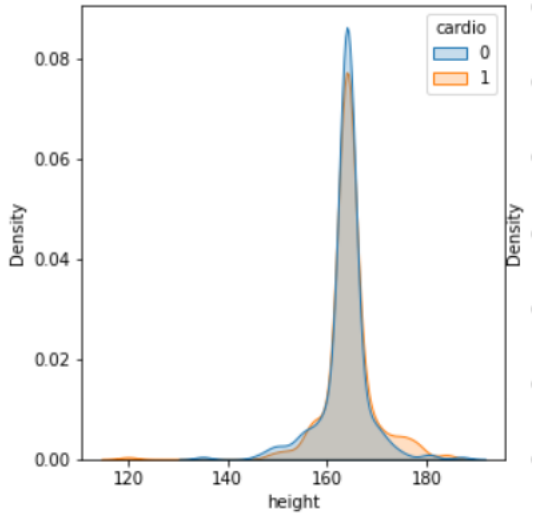
In the "gluc" kde plot, we can see that, people who have Normal(0) glucose have higher chance of cardio=0, i.e there is a high chance that they wont have cardiovascular disease, than the people who have more glucose.

Active vs cardio:



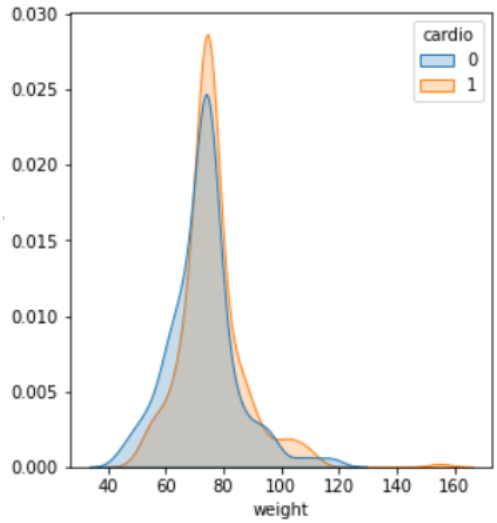
In the "active" kde plot, we can see that, people who are more physically active have a slightly higher chance of cardio = 0, i.e there is a high chance that they won't have cardiovascular disease, than the people who have less physically active.

Height vs cardio:



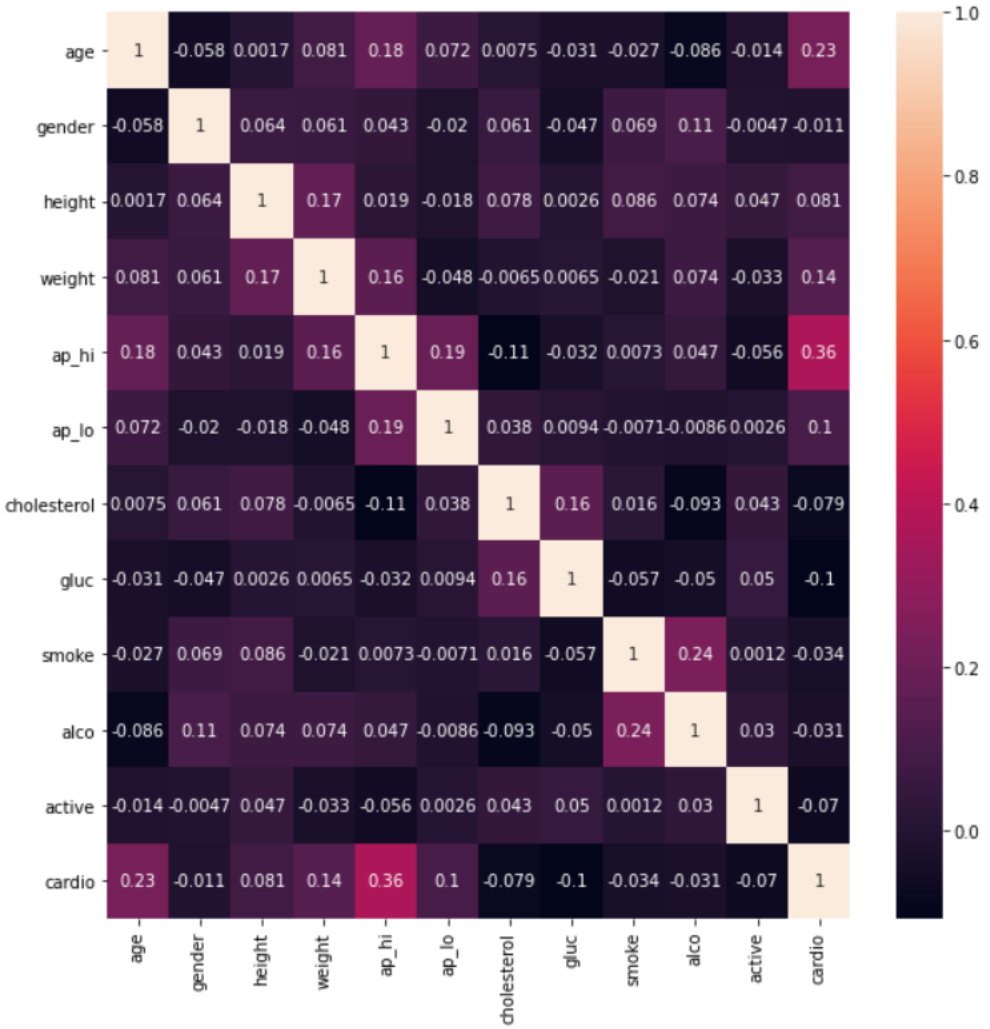
In kde plot of height, the plot of cardio=0 and cardio=1 is roughly the same and from this we can also tell height column does not add much importance, so we can drop height column when we are building our model.

Weight vs cardio:

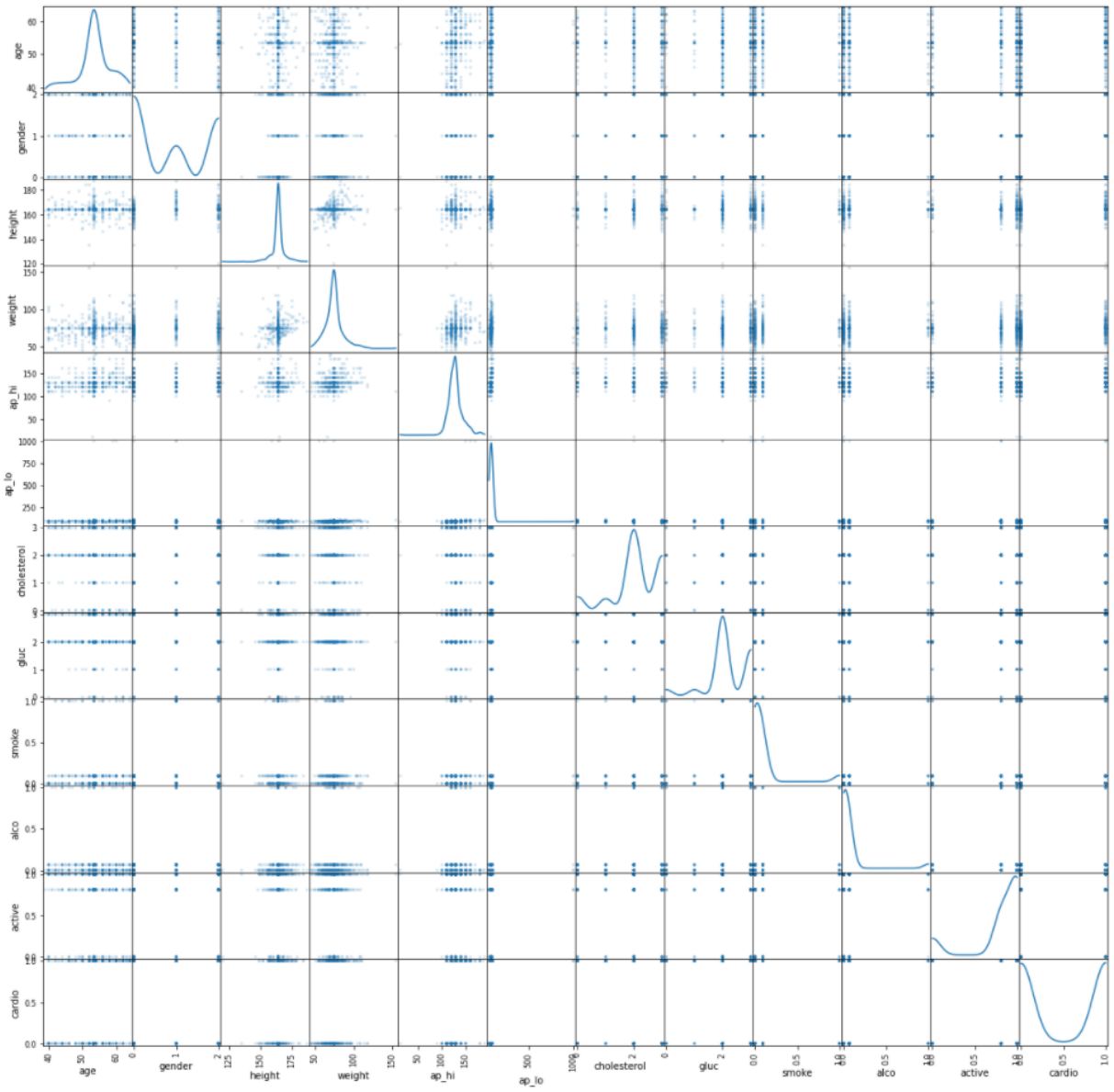


In kde plot of weight, the plot of cardio=0 and cardio=1 is roughly the same and from this we can also tell weight column does not add much importance, so we can drop weight column when we are building our model.

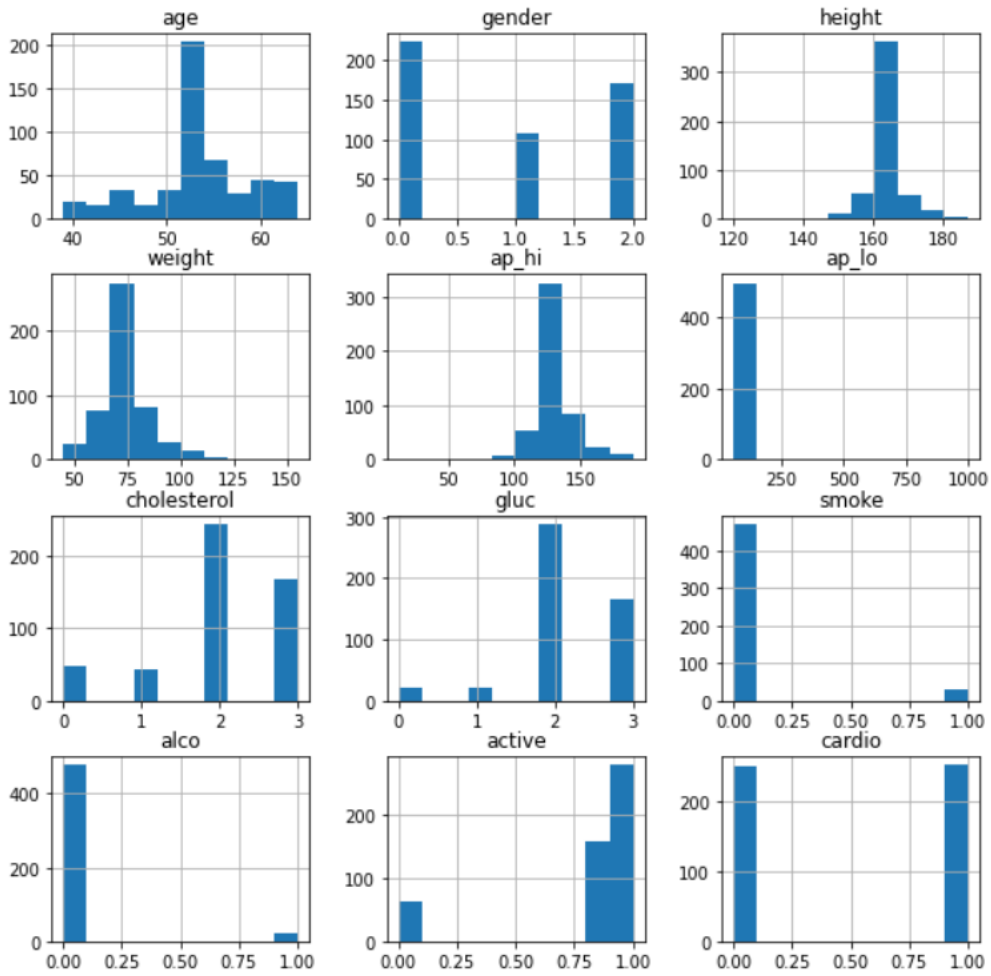
- PlotCorrelationMatrix – Plotting the correlation matrix, I can see correlations of each column with itself and other columns (This is a symmetric matrix). From the correlation matrix I can see that, “ap_hi”, “age” and “cholesterol” has the highest correlation with “cardio” when compared to other columns (shown below).



- PlotScatterMatrix – Plotting the Scatter Matrix, we can see the scatter matrix of each column with other column and we can see the kde plot of each column too. (shown below).



- PlotPerColumnDistribution – plots the histograms of each columns

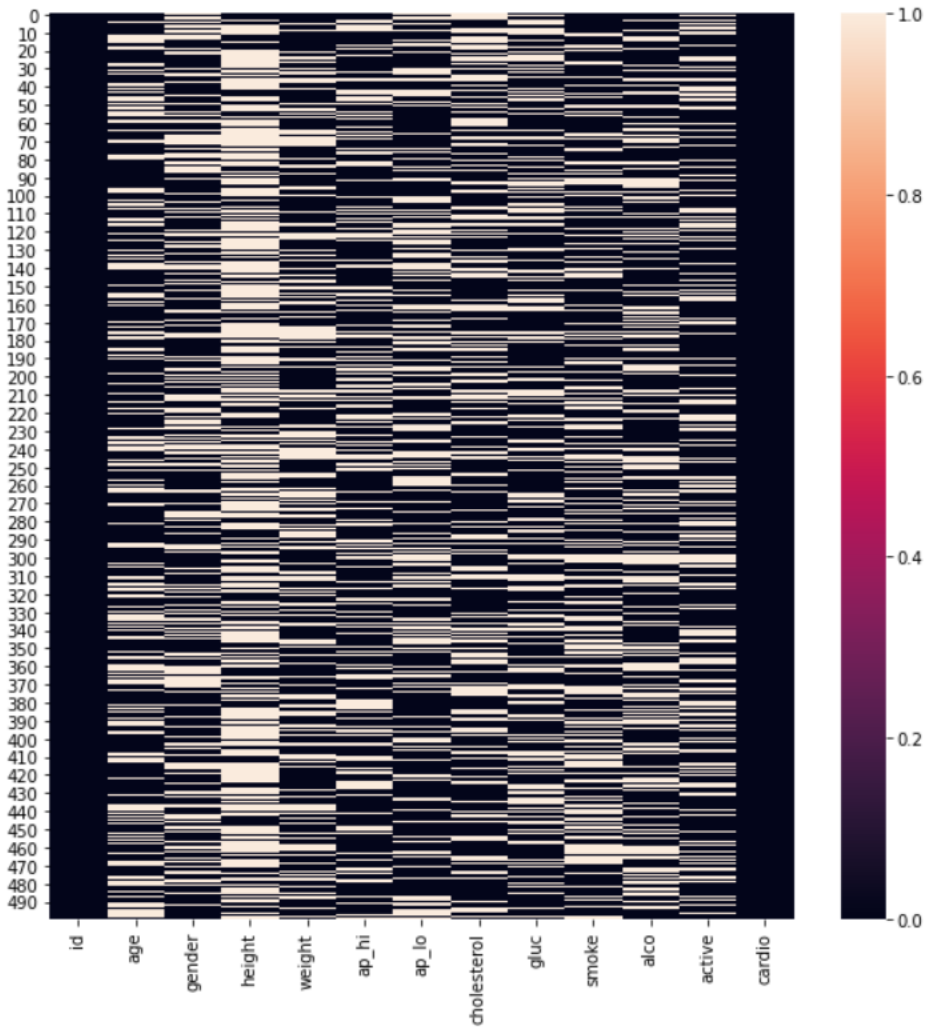


- Missing value imputation:
  I am loading the data again to try out different missing data imputation techniques.
  Just to recap, numbers of missing values in all columns are :

```
number of null values in column id: 0
number of null values in column age: 165
number of null values in column gender: 171
number of null values in column height: 302
number of null values in column weight: 164
number of null values in column ap_hi: 153
number of null values in column ap_lo: 168
number of null values in column cholesterol: 167
number of null values in column gluc: 167
number of null values in column smoke: 174
number of null values in column alco: 165
number of null values in column active: 157
number of null values in column cardio: 0
```

  Heatmap of missing values:
  Plotting the heatmap for missing values, In this plot, I can see that, The more the number of missing values in the column, the lighter the column is, for example, column "id" and column "cardio" have zero missing values, so they have a full dark shade,

meanwhile, height has the highest missing values, so it has a very light shade (shown below).



Different strategies of replacing NaN values:
All these strategies are evaluated by training and evaluating a LogisticRegression model.

i. Use dropna

```
Train Accuracy after dropping rows with NaN values:  1.0
Test Accuracy after dropping rows with NaN values:  0.5
f1 score after dropping rows with NaN values:  0.0
```

When I used dropna as an imputation strategy, I got Train accuracy as 1.0, Test accuracy as 0.5 and f1 score as 0, by looking at these numbers, I can say that the model is overfitting

ii. Use replace NA with zero or max value

```
Train Accuracy after replacing NaN with 0:  0.6375
Test Accuracy after replacing NaN with 0:  0.56
f1 score after replacing NaN with 0:  0.56
```

Replacing NaN with 0 gives test accuracy of 56%, which is improvement from dropna, but it is very bad accuracy for binary classification problem.

iii. Use replace NA with mean

```
Train Accuracy after replacing NaN with mean:  0.685
Test Accuracy after replacing NaN with mean:  0.68
f1 score after replacing NaN with mean:  0.6923076923076923
```

After replacing NaN values with mean of individual columns, we get test accuracy of 68% and f1 score of 69%. which is better result than previous 2 strategies. This method of replacement is better than dropna and replace with 0.

iv. Use replace NA with median

```
Train Accuracy after replacing NaN with median:  0.7
Test Accuracy after replacing NaN with median:  0.62
f1 score after replacing NaN with median:  0.6415094339622641
```

After replacing NaN values with median of individual columns, we get test accuracy of 62% and f1 score of 64%. Replacing NaN with mean is better strategy than replacing NaN with median.

v. Use replace NA with mode

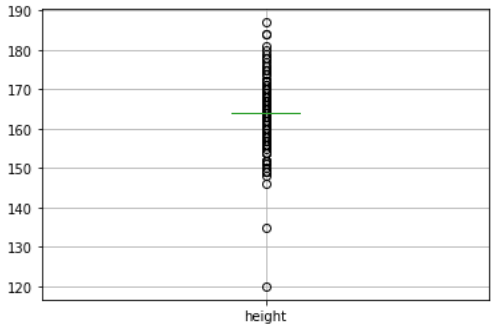
```
Train Accuracy after replacing NaN with median:  0.675
Test Accuracy after replacing NaN with median:  0.71
f1 score after replacing NaN with median:  0.7184466019417476
```

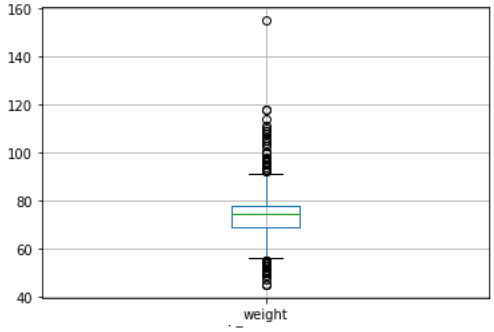
After replacing NaN values with mode of individual columns, we get test accuracy of 71% and f1 score of 71%. Replacing NaN with mode is better strategy than replacing NaN with mean or median.

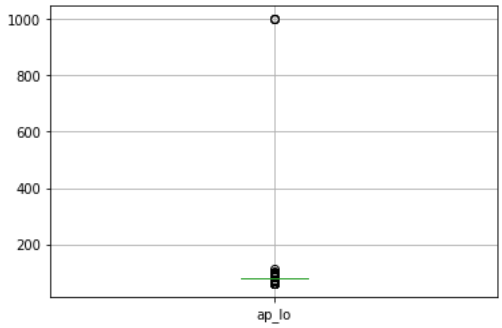
Building the Model:

Steps for preprocessing the data:

- Read cardio-train.csv and cardio-validation.csv to start fresh. We will use the knowledge acquired from EDA performed above to preprocess the data and do feature engineering to obtain best possible accuracy in our model.
- Drop “id” column as it is just identifier of each row and will not contribute in prediction task.
- Replace string values with integer in categorical columns like 'cholesterol', 'gluc', 'gender' with LabelEncoder.
- Divide age by 365 to convert it from days to years, which is easy to interpret.
- Replace NaN values in 'age', 'ap_hi' columns by mean of that column.
- Replace NaN values in 'ap_lo', 'height', 'weight' columns by median of that column since these columns have a lot of outliers as seen by boxplot.







- Replace NaN values in 'gender', 'cholesterol', 'gluc', 'smoke', 'alco', 'active' columns by mode of that column as they are categorical features.
- Outliers generally mean wrongly measured data. Since 'height', 'weight', 'ap_lo' columns have outliers as seen in boxplot above, it is a good idea to replace outliers with mean values before proceeding. Here I am considering values below 10% quantile and values above 90% quantile as outliers and replacing them with mean value which is 50% quantile.
- Calculate BMI feature from height, weight and drop height and weight columns. BMI is calculated as weight(kg)/height(meter)*height(meter)
- Use standard scaler to rescale values so that they have 0 mean and 1 std deviation after transformation. Scaling values before feeding the model makes the model robust against different columns having different value ranges.

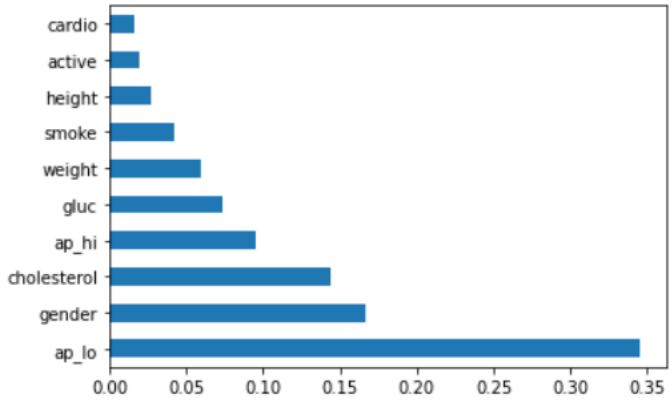
Training the models:

- First, I have tried making a handmade LogisticRegression Model. The custom logic of LogisticRegression implementation is as followed:

```
initializing the weights and bias to zero
for every epoch
    for every epoch
        for every data point(X_train,y_train)
          compute gradient w.r.to w
          compute gradient w.r.to b
          update w, b
        predict the output of x_train using w,b
        compute the loss between predicted and actual values
        store all the train loss values in a list
        predict the output of x_test using w,b
        compute the loss between predicted and actual values
        store all the test loss values in a list
        compare previous loss and current loss, if loss is not updating then
    stop the process and return w,b
```

  The parameters given to this model were: alpha=0.001, eta0(learning rate)=0.0001, epochs=8. This model gives 0.688 accuracy.

- Next is Logistic Regression model again, but I am using sklearn's implementation here. This model gives 0.716 accuracy and 0.705 f1 score. The parameters given to this model were C = 10, max_iter = 5000. Parameters were decided using hyperparameter tuning (loop of possible values).
- I also tried RandomForest model. First I tried to get the best possible hyperparameter values among these:
  estimators = [100,200,400]
  max_depth = range(5,12)
  min_samples_split = range(2,11)
  The best model was found for n_estimators = 100 max_depth = 6 min_samples_split = 9 with Accuracy = 0.744 and F1-score = 0.726
- RandomForest gives feature importances. I have plotted them here:



As we can see, ap_lo, gender and cholesterol are 3 most important features here.

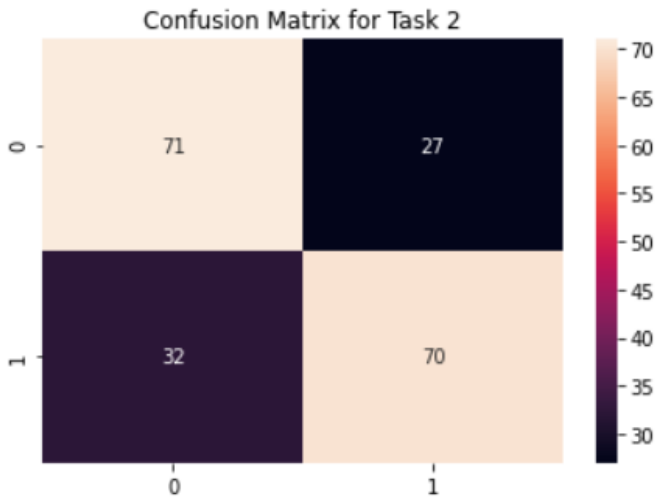
- For SVM model, I first tried to fetch best possible values of hyperparameters by running the model with following parameters:
  c_values = [10**x for x in range(-6,5)]
  kernel = ['poly', 'rbf', 'sigmoid','linear']
  gamma = ['auto']
  The best SVM model was for parameters kernel = linear, C = 10, gamma = auto, and gave Accuracy = 0.736, F1-score = 0.711.
- For K Nearest Neighbors alogirthm, I have tried passing every value between 1 to 100 as N-neighbors parameter. Best accuracy was achieved with n_neighbors = 55, Accuracy: 0.704, F1-score = 0.684.
- Next, I tried Bagging classifier, an ensemble method. I used default DecisionTree as base classifier. With n_estimators = 100, max_samples = 0.20, Bagging classifier gave Accuracy: 0.740, F1-score = 0.721.
- Next, I tried Voting classifier which collectively gives decisions based on Logistic Regression, KNN, SVM models trained above. Voting Classifier gave Accuracy = 0.732, F1-score = 0.712. Highest Kaggle score was achieved by this model.
- Since In Lab1 , allowed models were only SVM, Logistic Regression, Decision tree and Random forest, I will keep Random Forest model as my final model, which gives ~75% accuracy on local and 71% on kaggle.

Evaluating the models:

- For test data, we read cardio-test.csv data where all the features are given and we have to predict value of cardio column.
- We apply the same feature preprocessing which we applied to train data.
- We fit our Random Forest model with this data and save predictions in a csv file. This csv file is uploaded to Kaggle page to receive score.

Task 2

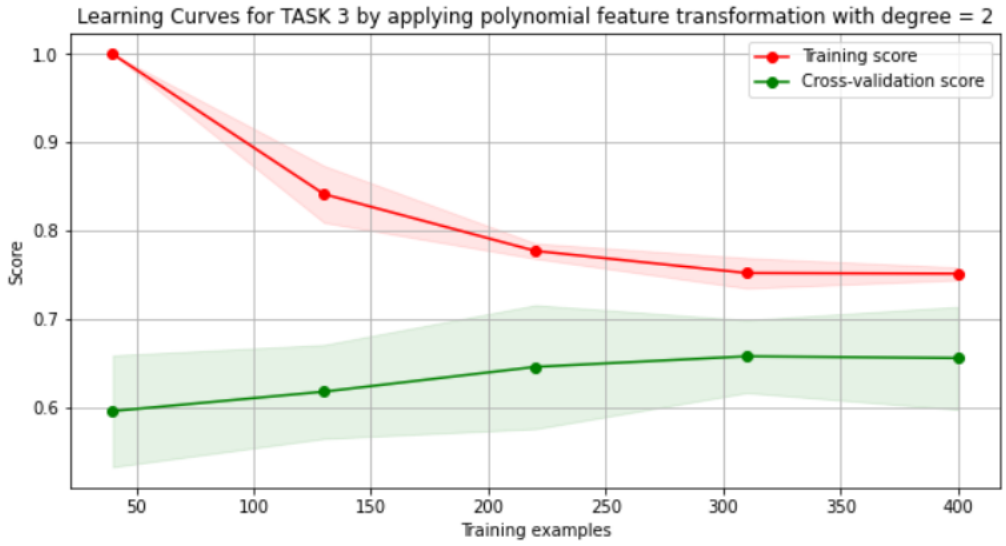
- In Task 2, It is asked to train the cardio-complete.csv and compare the models in task1 and task2.
- Complete data has 1000 rows while in task 1, training data had 500 rows and validation data had 500 rows. So in this part, we have more data to fit the model.
- The cardio complete data has 0 null values. Hence, we will require less preprocessing for this data. We do not need to worry about missing data imputation strategies for this task.
- I call same preprocess method used in Task 1 for this data. Which removes id column and divides age by 365. It also converts categorical features to numerical by using label encoder.
- Now I ran Logistic Regression model on this data. Logistic Regression Accuracy in Task 2= 0.705, F1-score = 0.704. This score was achieved by passing C = 0.01.
- Comparison of performance of LogisticRegression model on both task 1 and task 2 is summarized below:
  Model = Logistic Regression
  For Task 1 : Accuracy = 0.716, Precision = 0.720, Recall = 0.691, F1-score = 0.705
  For Task 2 : Accuracy = 0.705, Precision = 0.712, Recall = 0.686, F1-score = 0.694
- Next, I plot the confusion matrix of both models to gain more insights about TP, FP, TN, FN values.

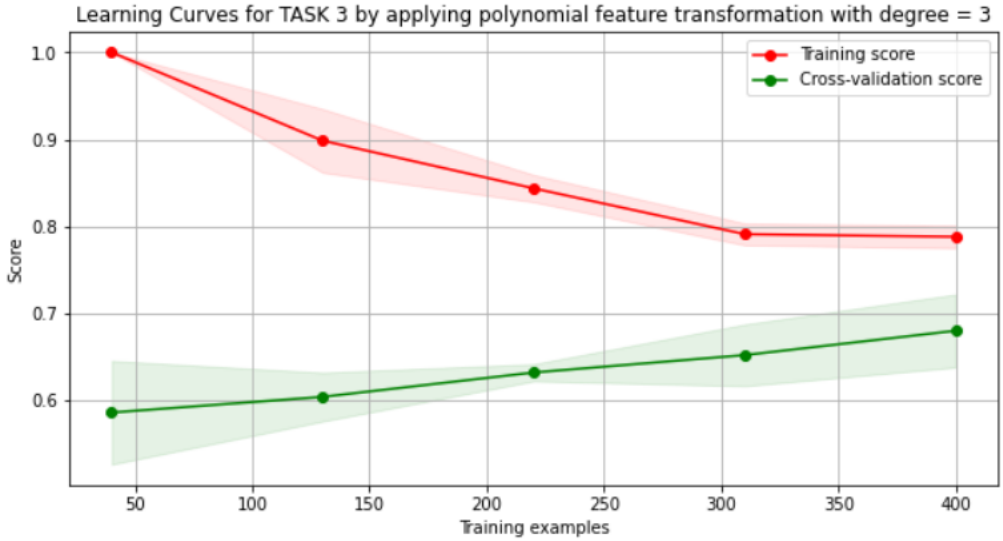


- From the above metrics, I can see that, the model I built on complete data (task-2) performs slightly poorly in terms of Accuracy, Precision, Recall and F1 score. This may be because of multiple reasons, one of the reasons might be that when I use complete data, I may also be incorporating noise in my data, which is making my model in task-2 perform poorly. In task 1 we intelligently use imputation methods, which may give better results than noisy data.
- From confusion matrix we can see that in task 1, 94 true records of class 0 are identified, and 85 true records of class 1 are identified. Both of these numbers are higher than model trained in task 2 (71 and 70 respectively). Thus, we can say that our model in task 1 is performing better.

Task 3

- For task 3, we had to apply feature transformation on train data. I have tried polynomial feature transformation with degree = 2 and degree = 3 and then compared the results of both.
- I am training LogisticRegression on train data after applying feature transformation.
- Logistic Regression Accuracy in Task 3 (degree = 2) = 0.692, F1-score = 0.662
  train accuracy with degree 2: 0.736
  test accuracy with degree 2: 0.692
- Logistic Regression Accuracy in Task 3 (degree = 3) = 0.672, F1-score = 0.647
  train accuracy with degree 3: 0.792
  test accuracy with degree 3: 0.672
- I call the plot_learning_curve method for both models to plot learning curve as shown below:





- I am using the plot_learning_curve function from https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html
- Lets look at the training and testing accuracies achieved by transforming the data with degree 2 and 3 to compare if varying polynomial degree helps us with performance of the model.
- train accuracy with degree 2: 0.736
  test accuracy with degree 2: 0.692
  train accuracy with degree 3: 0.792
  test accuracy with degree 3: 0.672
- Increasing polynomial degree does improve the train accuracy but at the cost of cross-validation accuracy which then leads to overfitting.
- Varying polynomial degree does not improve cross-validation (test) accuracy, in fact after a certain degree value, cross-validation (test) accuracy starts decreasing and train accuracy starts increasing leading to overfitting.