

Summary Diabetes_Project

October 13, 2020

1 Summary Notebook

- An overview of project to provide reader with quick and brief insight of project.

2 Research Question

Using Logistic Regression can a machine learning model accurately predict whether or not the patients in the dataset have diabetes or not?

2.1 Data Dictionary:

Column Name	Description
Pregnancies	Number of times pregnant
Glucose	Plasma Glucose level Oral Glucose Test
Blood Pressure	Diastolic Blood pressure in mmHg
Skin Thickness	Triceps skin fold thickness in mm
Insulin.	2-Hour insulin level in mU/ml
BMI	Body Mass Index in Kg
Diabetes Pedegree Func.	Diabetes Pedegree Function
Age	Age (years)
Outcome	Class Variable (0 or 1)

2.1.1 Summery Statistics

We can see outliers in summery statistics. In pregnancy, 75% of the data lies within 6. Therefore when we see a max of 17 we know that is an outlier. In insulin we can also conclude the same thing, as 75% of the data is within 127.25, therefore 846 is also considered an outlier.

```
[60]: diabetes.describe().style.set_caption('There are 768 rows, with potential_
      ↪outliers.')
```

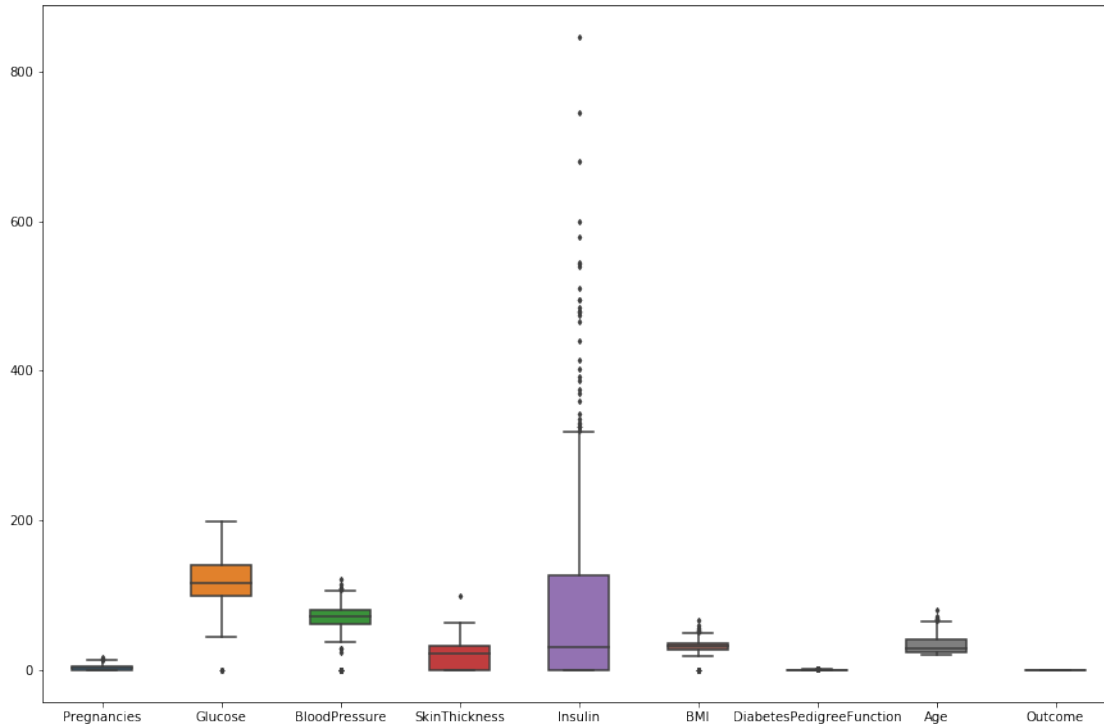
```
[60]: <pandas.io.formats.style.Styler at 0x7fb12d505d50>
```

2.2 Visualizing Outliers:

Below a box and whisker plot depicts what was summerized in the summery statistics. As one can denote the numerous outliers that are portrayed in the insulin class.

```
[90]: fig, ax = plt.subplots(figsize=(15,10))
      sns.boxplot(data=diabetes, width= 0.5,ax=ax, fliersize=3)
```

```
[90]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb12e0d7250>
```

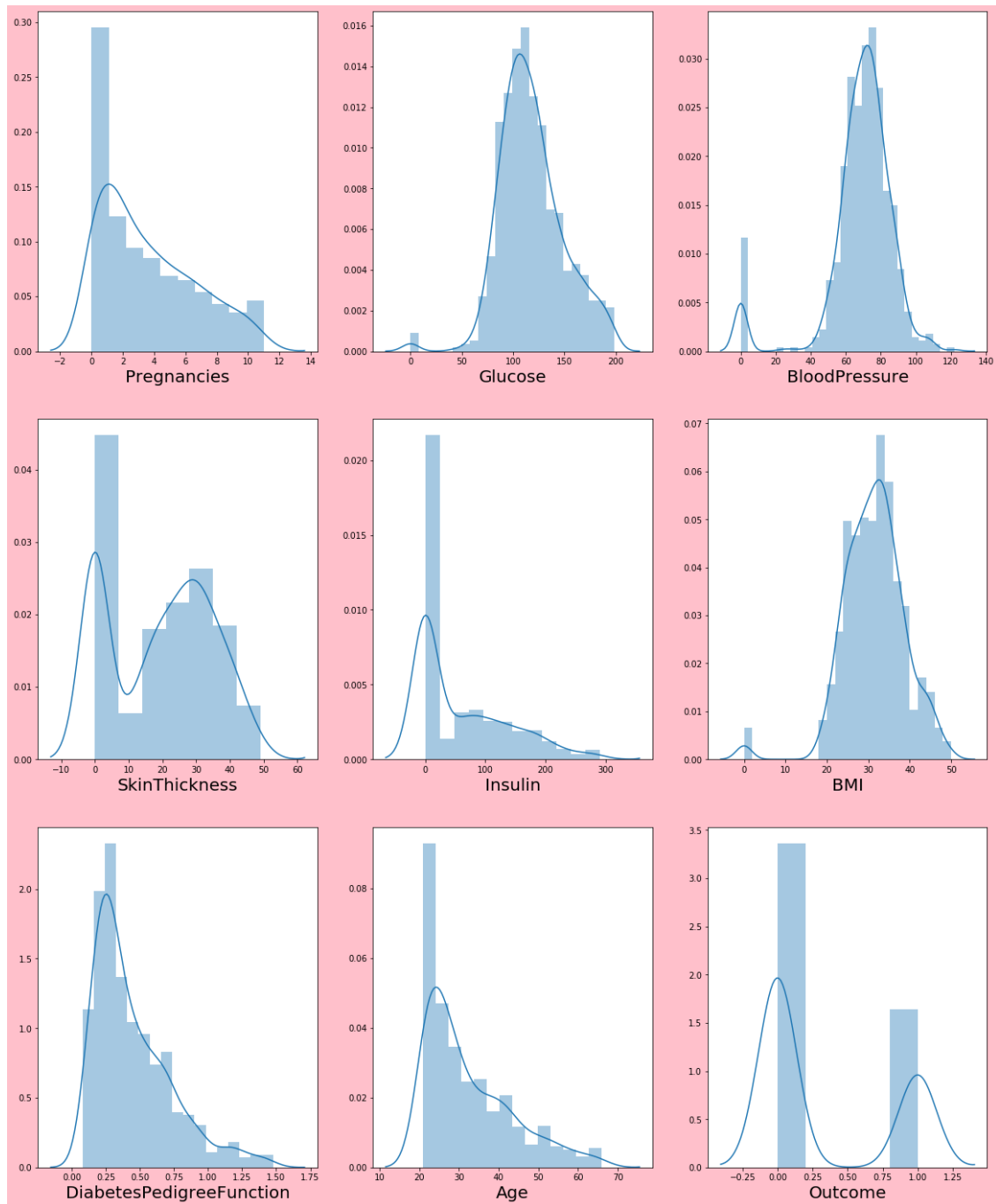


2.2.1 Revisiting Skewness after Outlier handling

The data looks much better now than before. We will start our analysis with this data now as we don't want to lose important information. If our model doesn't work with accuracy, we will come back for more preprocessing.

```
[65]: plt.figure(figsize=(20,25), facecolor='pink')
      plotnumber = 1

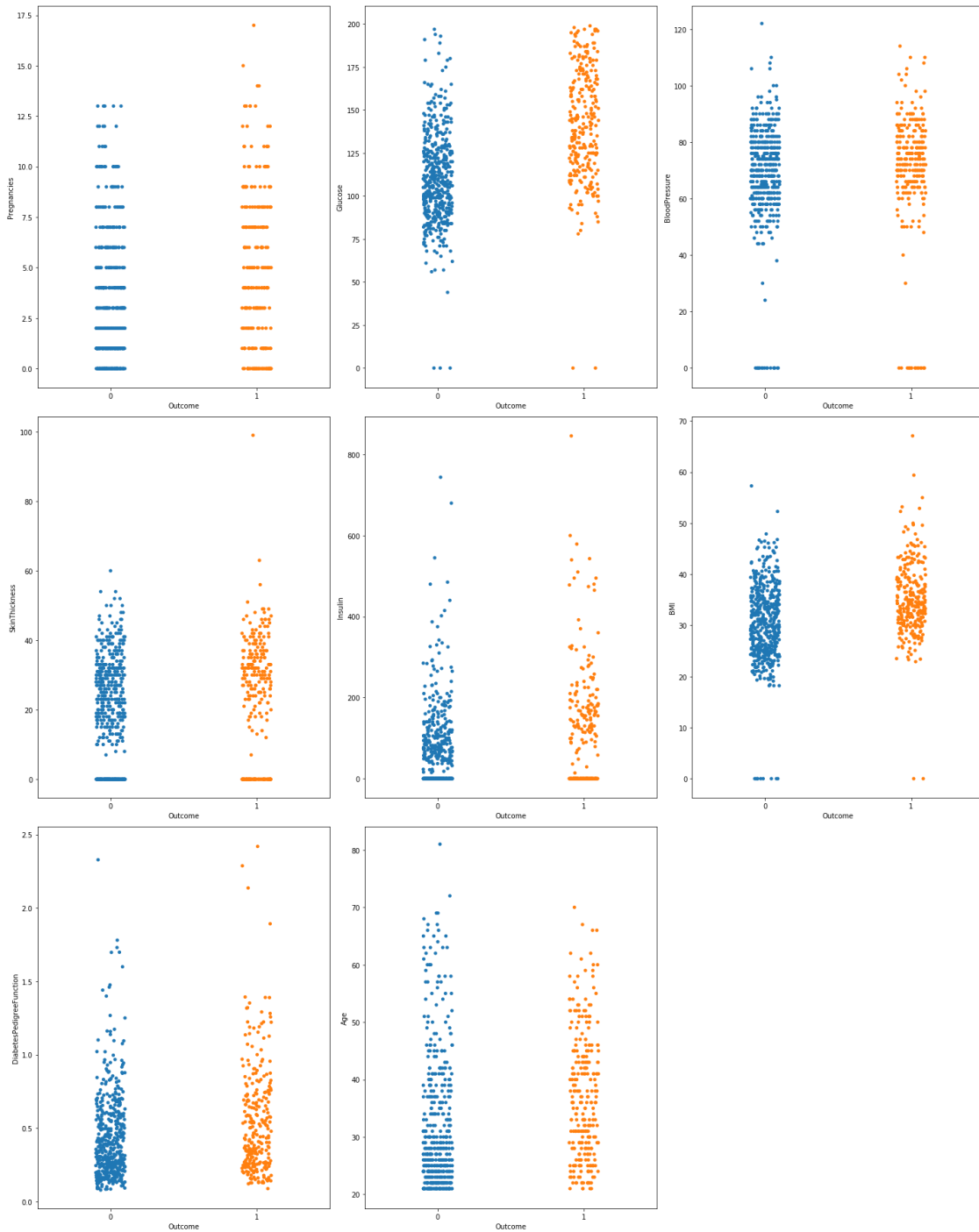
      for column in data_cleaned:
          if plotnumber<=9 :
              ax = plt.subplot(3,3,plotnumber)
              sns.distplot(data_cleaned[column])
              plt.xlabel(column,fontsize=20)
              #plt.ylabel('Salary',fontsize=20)
              plotnumber+=1
      plt.show()
```



```
[67]: plt.figure(figsize=(20,25), facecolor='white')
      plotnumber = 1

      for column in X:
          if plotnumber<=9 :
              ax = plt.subplot(3,3,plotnumber)
```

```
sns.stripplot(y,X[column])
plotnumber+=1
plt.tight_layout()
```



2.3 Scale features

Let's proceed by checking multicollinearity in the dependent variables. Before that, we should scale our data. Let's use the standard scaler for that.

```
[68]: scalar = StandardScaler()
      X_scaled = scalar.fit_transform(X)
```

This is how our data looks now after scaling. Great, now we will check for multicollinearity using VIF(Variance Inflation factor)

```
[69]: X_scaled
```

```
[69]: array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
              0.46849198,  1.4259954 ],
            [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
              -0.36506078, -0.19067191],
            [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
              0.60439732, -0.10558415],
            ...,
            [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
              -0.68519336, -0.27575966],
            [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
              -0.37110101,  1.17073215],
            [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
              -0.47378505, -0.87137393]])
```

3 Model Building

3.1 Split Test and Train data

Split our data into training data and testing data to test model. Random state utilized to ensure consistent values.

```
[71]: x_train,x_test,y_train,y_test = train_test_split(X_scaled,y, test_size= 0.25,
      ↪random_state = 355433333)
```

```
[72]: log_reg = LogisticRegression()

      log_reg.fit(x_train,y_train)
```

```
[72]: LogisticRegression()
```

4 Model Results and Performance

Below are the probabilities of each outcome. If probabilities. >50% = class 0, if <50% = class 1. Therefore by looking at the probabilities one could denote for example 0.88, 0.65, .90 would be class 0. The next one 0.25 would fall into class 1.

```
[91]: log_reg.predict_proba(x_test)
```

```
[91]: array([[0.88743777, 0.11256223],
             [0.65294724, 0.34705276],
             [0.9012626 , 0.0987374 ],
             [0.25085199, 0.74914801],
             [0.96429361, 0.03570639],
             [0.80294778, 0.19705222],
             [0.94339524, 0.05660476],
             [0.52570717, 0.47429283],
             [0.83610742, 0.16389258],
             [0.97430986, 0.02569014],
             [0.8855256 , 0.1144744 ],
             [0.35728758, 0.64271242],
             [0.59218068, 0.40781932],
             [0.73555962, 0.26444038],
             [0.90855784, 0.09144216],
             [0.79109229, 0.20890771],
             [0.30785064, 0.69214936],
             [0.77915586, 0.22084414],
             [0.50295763, 0.49704237],
             [0.90462927, 0.09537073],
             [0.92791088, 0.07208912],
             [0.2982658 , 0.7017342 ],
             [0.45542995, 0.54457005],
             [0.68346221, 0.31653779],
             [0.76609034, 0.23390966],
             [0.00448421, 0.99551579],
             [0.3853186 , 0.6146814 ],
             [0.72437731, 0.27562269],
             [0.61502056, 0.38497944],
             [0.81300099, 0.18699901],
             [0.47558432, 0.52441568],
             [0.61220105, 0.38779895],
             [0.54689147, 0.45310853],
             [0.63220662, 0.36779338],
             [0.9783213 , 0.0216787 ],
             [0.93753267, 0.06246733],
             [0.11534178, 0.88465822],
             [0.91841242, 0.08158758],
             [0.91721364, 0.08278636],
             [0.73630904, 0.26369096],
             [0.82332655, 0.17667345],
             [0.554024 , 0.445976 ],
             [0.28378254, 0.71621746],
             [0.40151676, 0.59848324],
             [0.92721322, 0.07278678],
```

[0.92909081, 0.07090919],
[0.71490005, 0.28509995],
[0.1019511 , 0.8980489],
[0.81955029, 0.18044971],
[0.6952575 , 0.3047425],
[0.90110799, 0.09889201],
[0.93963441, 0.06036559],
[0.52513164, 0.47486836],
[0.28333969, 0.71666031],
[0.95292226, 0.04707774],
[0.67700711, 0.32299289],
[0.25166792, 0.74833208],
[0.8743582 , 0.1256418],
[0.71875871, 0.28124129],
[0.72326145, 0.27673855],
[0.64371508, 0.35628492],
[0.88409326, 0.11590674],
[0.2789992 , 0.7210008],
[0.53016429, 0.46983571],
[0.97965985, 0.02034015],
[0.47003707, 0.52996293],
[0.25444639, 0.74555361],
[0.67208573, 0.32791427],
[0.23562452, 0.76437548],
[0.67788611, 0.32211389],
[0.22025458, 0.77974542],
[0.60407926, 0.39592074],
[0.79641852, 0.20358148],
[0.76641868, 0.23358132],
[0.96159523, 0.03840477],
[0.23992256, 0.76007744],
[0.96399773, 0.03600227],
[0.2456378 , 0.7543622],
[0.94254906, 0.05745094],
[0.1112945 , 0.8887055],
[0.71955195, 0.28044805],
[0.91112984, 0.08887016],
[0.80916991, 0.19083009],
[0.70411416, 0.29588584],
[0.89857929, 0.10142071],
[0.6855641 , 0.3144359],
[0.29950176, 0.70049824],
[0.69610573, 0.30389427],
[0.14339442, 0.85660558],
[0.53805651, 0.46194349],
[0.57642335, 0.42357665],
[0.63755467, 0.36244533],

[0.85086242, 0.14913758],
[0.94321319, 0.05678681],
[0.95006643, 0.04993357],
[0.80524173, 0.19475827],
[0.84133375, 0.15866625],
[0.04657625, 0.95342375],
[0.86429293, 0.13570707],
[0.93703968, 0.06296032],
[0.80536885, 0.19463115],
[0.04131887, 0.95868113],
[0.6734605 , 0.3265395],
[0.82717142, 0.17282858],
[0.29055759, 0.70944241],
[0.26428764, 0.73571236],
[0.67992262, 0.32007738],
[0.8733264 , 0.1266736],
[0.76518384, 0.23481616],
[0.78260574, 0.21739426],
[0.78499239, 0.21500761],
[0.85524255, 0.14475745],
[0.56031726, 0.43968274],
[0.69134523, 0.30865477],
[0.63777053, 0.36222947],
[0.66775124, 0.33224876],
[0.8301115 , 0.1698885],
[0.33012308, 0.66987692],
[0.73452606, 0.26547394],
[0.91376478, 0.08623522],
[0.41173915, 0.58826085],
[0.8196284 , 0.1803716],
[0.77377145, 0.22622855],
[0.81559602, 0.18440398],
[0.84774314, 0.15225686],
[0.89538474, 0.10461526],
[0.12286904, 0.87713096],
[0.90208783, 0.09791217],
[0.92944673, 0.07055327],
[0.69692684, 0.30307316],
[0.91062558, 0.08937442],
[0.64058389, 0.35941611],
[0.32304546, 0.67695454],
[0.96768994, 0.03231006],
[0.89934202, 0.10065798],
[0.81288678, 0.18711322],
[0.96279501, 0.03720499],
[0.2111741 , 0.7888259],
[0.95708512, 0.04291488],

[0.75443945, 0.24556055],
[0.19555231, 0.80444769],
[0.88863717, 0.11136283],
[0.86836881, 0.13163119],
[0.76715824, 0.23284176],
[0.65201173, 0.34798827],
[0.06387934, 0.93612066],
[0.463008 , 0.536992],
[0.94588587, 0.05411413],
[0.40896707, 0.59103293],
[0.76321829, 0.23678171],
[0.86805431, 0.13194569],
[0.53031738, 0.46968262],
[0.16601158, 0.83398842],
[0.82789632, 0.17210368],
[0.82742288, 0.17257712],
[0.71934289, 0.28065711],
[0.84220951, 0.15779049],
[0.6871105 , 0.3128895],
[0.94503822, 0.05496178],
[0.05377321, 0.94622679],
[0.62905404, 0.37094596],
[0.53696637, 0.46303363],
[0.46450285, 0.53549715],
[0.95737304, 0.04262696],
[0.80124225, 0.19875775],
[0.95443773, 0.04556227],
[0.89715688, 0.10284312],
[0.26126122, 0.73873878],
[0.57796452, 0.42203548],
[0.23387119, 0.76612881],
[0.92153971, 0.07846029],
[0.6955245 , 0.3044755],
[0.29521694, 0.70478306],
[0.2060838 , 0.7939162],
[0.58953471, 0.41046529],
[0.58675013, 0.41324987],
[0.79142046, 0.20857954],
[0.93996904, 0.06003096],
[0.84747151, 0.15252849],
[0.92448845, 0.07551155],
[0.90898274, 0.09101726],
[0.81487979, 0.18512021],
[0.27762498, 0.72237502],
[0.69887482, 0.30112518],
[0.87691381, 0.12308619],
[0.74794965, 0.25205035],

```
[0.96020777, 0.03979223],
[0.14114736, 0.85885264],
[0.93202331, 0.06797669],
[0.77280888, 0.22719112],
[0.91902486, 0.08097514],
[0.92910556, 0.07089444]])
```

4.1 Accuracy

Accuracy supports the predictions of the model against original values. Accuracy metrics help us determine or predict the true diabetic person out of the population. It is the fraction of predicted diabetic patients that the model got right.

```
[73]: y_pred = log_reg.predict(x_test)
```

Prediction of either class 0 or class 1 depending on the values above if they are less or more than 50%.

```
[85]: y_pred
```

```
[85]: array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
          1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
          0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
          1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
          1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
          1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
          0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
          0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0])
```

```
[74]: accuracy = accuracy_score(y_test,y_pred)
accuracy
```

```
[74]: 0.75
```

4.2 Confusion Matrix

The confusion matrix describes the performance of the classification model, which compares predicted classes to original/actual classes.

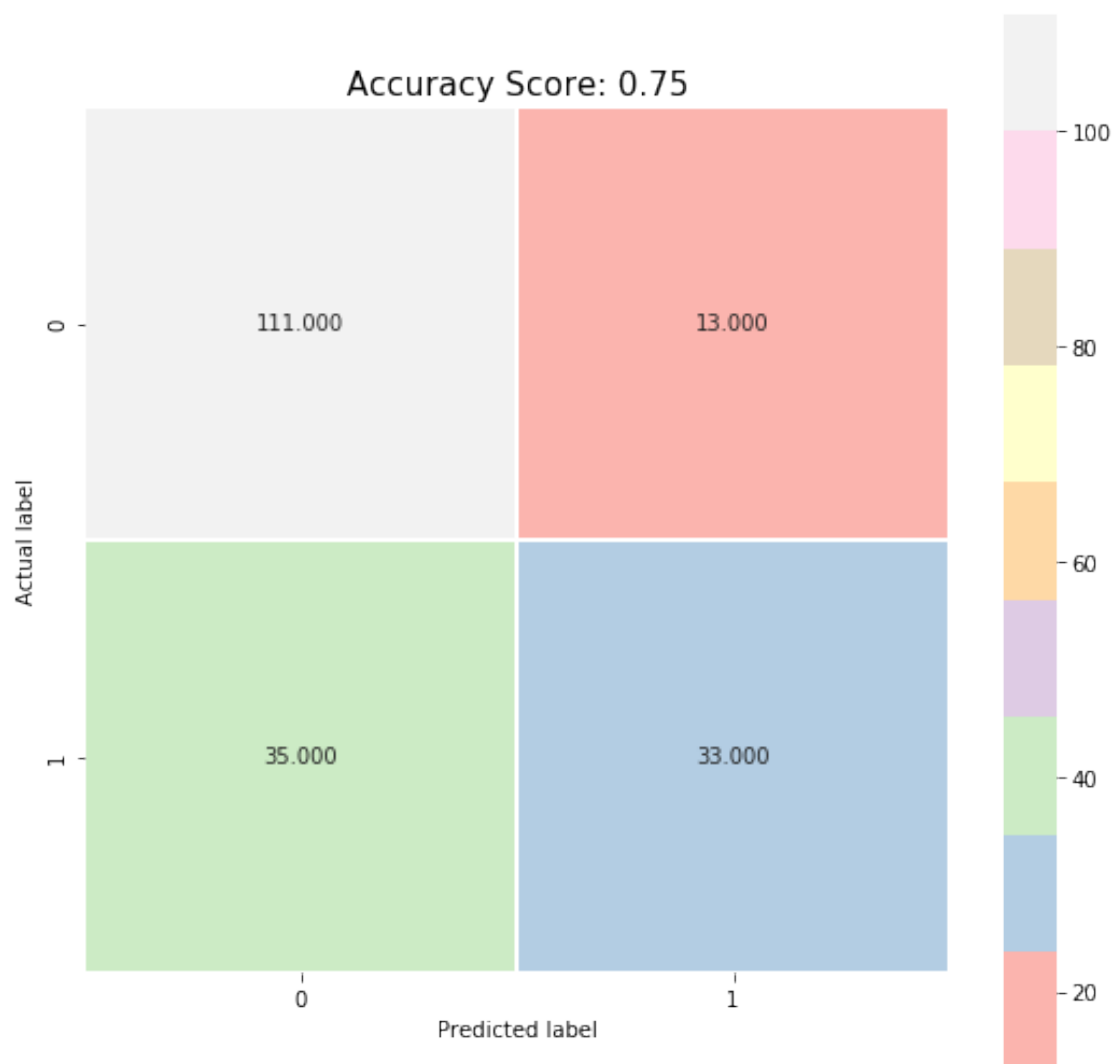
The summery of number of correct and incorrect predictions with count values segregated by classes.

```
[75]: conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

```
[75]: array([[111, 13],
          [ 35, 33]])
```

This confusion matrix shows that: - 111 are True positive meaning the actual class was the same as the predicted class. - 33 are True negative meaning the actual and predicted class matched.

```
[96]: plt.figure(figsize=(9,9))
sns.heatmap(conf_mat, annot=True, fmt=".3f", linewidths=.5, square = True, cmap=
    ↪= 'Pastell1');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(accuracy)
plt.title(all_sample_title, size = 15);
plt.savefig('toy_Digits_ConfusionSeabornCodementor.png')
plt.show()
```



Another way to quantitatively interpret the confusion matrix.

4.3 Precision

What are the relevant (what you have predicted and actual are matching.) positive diabetic patients out of entire positive diabetic predictions.

```
[78]: # Precision
Precision = true_positive/(true_positive+false_positive)
Precision
```

```
[78]: 0.8951612903225806
```

4.4 Recall

Fraction of actual diabetic patients to the number of diabetic patients retrieved from population.

```
[79]: # Recall
Recall = true_positive/(true_positive+false_negative)
Recall
```

```
[79]: 0.7602739726027398
```

4.5 F1_Score

Harmonic mean of precision and recall. We consider F1_score because it balances between precision and recall.

```
[80]: # F1 Score
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

```
[80]: 0.8222222222222222
```

4.6 Area Under Curve

Measure for a model how it can distinguish two separate groups under the target variable. The more area under the curve the better the model. Area under curve should be high. Blue is threshold.

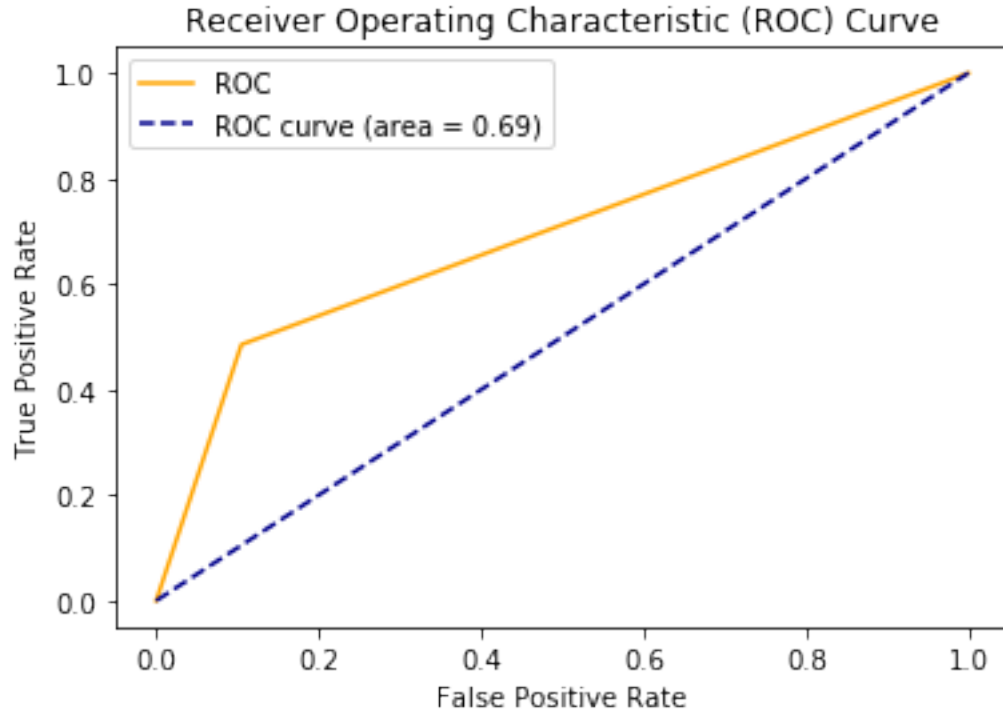
```
[81]: # Area Under Curve
auc = roc_auc_score(y_test, y_pred)
auc
```

```
[81]: 0.6902277039848197
```

```
[82]: fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

```
[83]: plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve_
→(area = %0.2f)' % auc)
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



5 Conclusion Summary

Logistic regression is a regression algorithm which can be used for performing classification problems. It calculates the probability that a given value belongs to a specific class. In this particular study we utilized the diabetes dataset to determine whether the features could accurately predict whether a patient would have diabetes. After modeling our data we were able to produce convincing results that the features in this dataset do in fact increase the odds and the probability of a patient being diagnosed with diabetes.