Multilinear Regression Model

 # importing train_test_split from sklearn
from sklearn.model_selection import train_test_split

#Assign the input feature to X and target class to Y

X = df.drop('diabetes', axis = 1)
Y = df['diabetes']

 # splitting the data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)

 # importing module
from sklearn.linear_model import LinearRegression

 # creating an object of LinearRegression class
mlr = LinearRegression()

 # fitting the training data
mlr.fit(x_train,y_train)

#Intercept and Coefficientprint("Intercept: ", mlr.intercept_)
print("Coefficients:")
list(zip(x, mlr.coef_))

#Prediction of test set
y_pred_mlr= mlr.predict(x_test)
#Predicted values
print("Prediction for test set: {}".format(y_pred_mlr))

```
#Actual value and the predicted value
mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_mlr})
slr_diff.head()

#Model Evaluation
from sklearn import metrics

meanAbErr = metrics.mean_absolute_error(y_test, y_pred_mlr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_mlr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_mlr))

print('R squared: {:.2f}'.format(mlr.score(x,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

Note

R Square is the coefficient of determination. It tells us how many points fall on the regression line. If the value of R Square is 90.11, which indicates that 90.11% of the data fit the regression model.

Mean Absolute Error is the absolute difference between the actual or true values and the predicted values. The lower the value, the better is the model’s performance. If the mean absolute error is 0, it means that your model is a perfect predictor of the outputs. If the the mean absolute error obtained is 1.227, it is pretty good as it is close to 0.

Mean Square Error is calculated by taking the average of the square of the difference between the original and predicted values of the data. The lower the value, the better is the model's performance. If the mean square error obtained is 2.636, it is pretty good.

Root Mean Square Error is the standard deviation of the errors which occur when a prediction is made on a dataset. This is the same as Mean Squared Error, but the root of the value is considered while determining the accuracy of the model. The lower the value, the better is the model's performance. If the root mean square error obtained for this particular model is 1.623, it is pretty good.

Logistic Regression

#Assign the input feature to X and target class to Y

```
X = df.drop('diabetes', axis = 1)
Y = df['diabetes']
```

Split the dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
   X, Y, test_size=0.25, random_state=42)
```

 # import the class

```
from sklearn.linear_model import LogisticRegression
```

 # instantiate the model (using the default parameters)

```
logreg = LogisticRegression(random_state=16)
```

 # fit the model with data

```
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)
```

 # Model evaluation using confusion matrix. import the metrics class

```
from sklearn import metrics

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

visualize the confusion matrix using Heatmap.

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Confusion matrix evaluation metrics

```
from sklearn.metrics import classification_report
target_names = ['0', '1']
print(classification_report(y_test, y_pred,
target_names=target_names))
```

```
          precision    recall  f1-score   support

0       0.79      0.93      0.86       123
 1       0.83      0.57      0.67        69

      accuracy                         0.80       192
    macro avg       0.81      0.75      0.77       192
  weighted avg       0.81      0.80      0.79       192
```

ROC curve

Receiver Operating Characteristic (ROC) curve is a plot of the true positive rate against the false positive rate. It shows the tradeoff between sensitivity and specificity.

```
y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```