ELEN 520 PROJECT REPORT SUBMISSION LAURENCE KIM

1.

Null Hypothesis: There is no relationship between the response variable and the 8 predictors.
Alternative Hypothesis: There is a relationship between the response variable and the 8 predictors.

2.

All the import and include functions are as follows in this section below}

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.graphics.regressionplots import influence_plot
from statsmodels.stats.outliers_influence import OLSInfluence
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import LogisticRegression,LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RidgeCV
```

Code Segment: Import and include libraries

Next, I set up my experiment to take in a csv called diabetes.csv which is in the same directory as my current python file I am running called project.py

The read function is as shows.

```
# Load the diabetes dataset
df = pd.read_csv('diabetes.csv')
print(df)
```

Code Segment: Read CSV function

The output of the print for this df is as shows in the code section below

```
    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
DiabetesPedigreeFunction  Age  Outcome
0            6      148             72             35        0  33.6                     0.627   50        1
1            1       85             66             29        0  26.6                     0.351   31        0
2            8      183             64              0        0  23.3                     0.672   32        1
3            1       89             66             23       94  28.1                     0.167   21        0
4            0      137             40             35      168  43.1                     2.288   33        1
..         ...      ...            ...            ...      ...   ...                       ...  ...      ...
763         10      101             76             48      180  32.9                     0.171   63        0
764          2      122             70             27        0  36.8                     0.340   27        0
765          5      121             72             23      112  26.2                     0.245   30        0
766          1      126             60              0        0  30.1                     0.349   47        1
767          1       93             70             31        0  30.4                     0.315   23        0

[768 rows x 9 columns]
```

Terminal Output: Print dataframe

This csv can be visualized as 767 different rows of data plots with 8 different predictors and the 'outcome' as the set that we are trying to predict. I also set up a training and test set based on a 80/20 split as shown in this code below.

```python
# Split the dataset into features (X) and target (y)
    columns_selected = "+".join(df.columns.difference(["Outcome"]))
    my_formula = "Outcome~" + columns_selected


    X = df.drop('Outcome', axis=1)
    y = df['Outcome']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

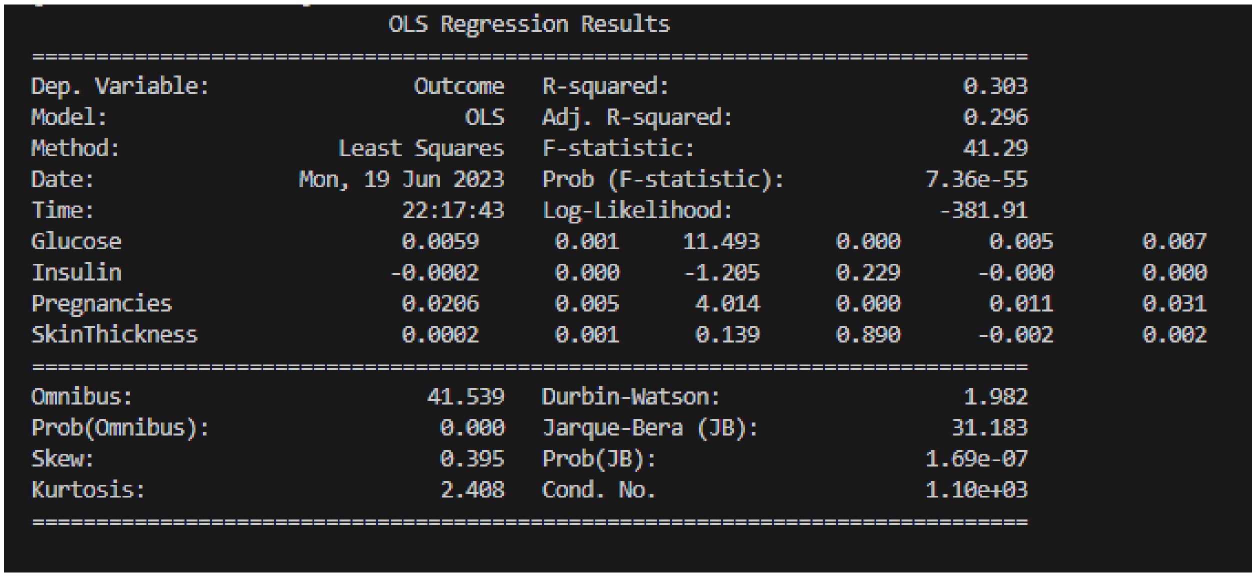
Code Segment: Dataset Restructuring

3.
This part of the code is pretty much the same for both models on which I am testing this dataset on. The first model I am building is a multi-linear regression model while the other model is a logistic regression model. I have included both model function definitions below as seen in the segments of code.

```python
# Function to perform multiple linear regression and evaluate the model
def mlRegression(df,k):
    # Split the dataset into features (X) and target (y)
    columns_selected = "+".join(df.columns.difference(["Outcome"]))
    my_formula = "Outcome~" + columns_selected


    X = df.drop('Outcome', axis=1)
    y = df['Outcome']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Create a linear regression model
    model = LinearRegression()
    lm_fit = smf.ols(my_formula, data=df).fit()
    print(lm_fit.summary())
    lm_fit.resid.describe()

    # Fit the model to the training data
    model.fit(X_train, y_train)

    # Predict the target values for the test data
    y_pred = model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print('Mean Squared Error:', mse)
    print('R-squared Score:', r2)
```

Code Segment: Multi linear model generation

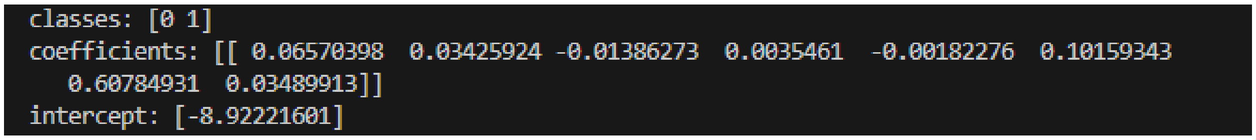
For the multi regression model which we have created, I have created some print functions that give us a summary of the model based at statsmodel.api’s summary function, as well as some parameters that would be useful in interpreting the model like R-squared, mean squared error, and R-squared. I create two models, one using the sklearn library and one using the statsmodel library. The manual prints that I create are that of the sklearn library build that only uses the training sets, while the sk learn uses the entire data frame.



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Outcome   R-squared:                       0.303
Model:                            OLS   Adj. R-squared:                  0.296
Method:                 Least Squares   F-statistic:                     41.29
Date:                Mon, 19 Jun 2023   Prob (F-statistic):           7.36e-55
Time:                        22:17:43   Log-Likelihood:                -381.91
Glucose                        0.0059      0.001     11.493      0.000       0.005       0.007
Insulin                       -0.0002      0.000     -1.205      0.229      -0.000       0.000
Pregnancies                    0.0206      0.005      4.014      0.000       0.011       0.031
SkinThickness                  0.0002      0.001      0.139      0.890      -0.002       0.002
==============================================================================
Omnibus:                       41.539   Durbin-Watson:                   1.982
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               31.183
Skew:                           0.395   Prob(JB):                     1.69e-07
Kurtosis:                       2.408   Cond. No.                     1.10e+03
==============================================================================
```

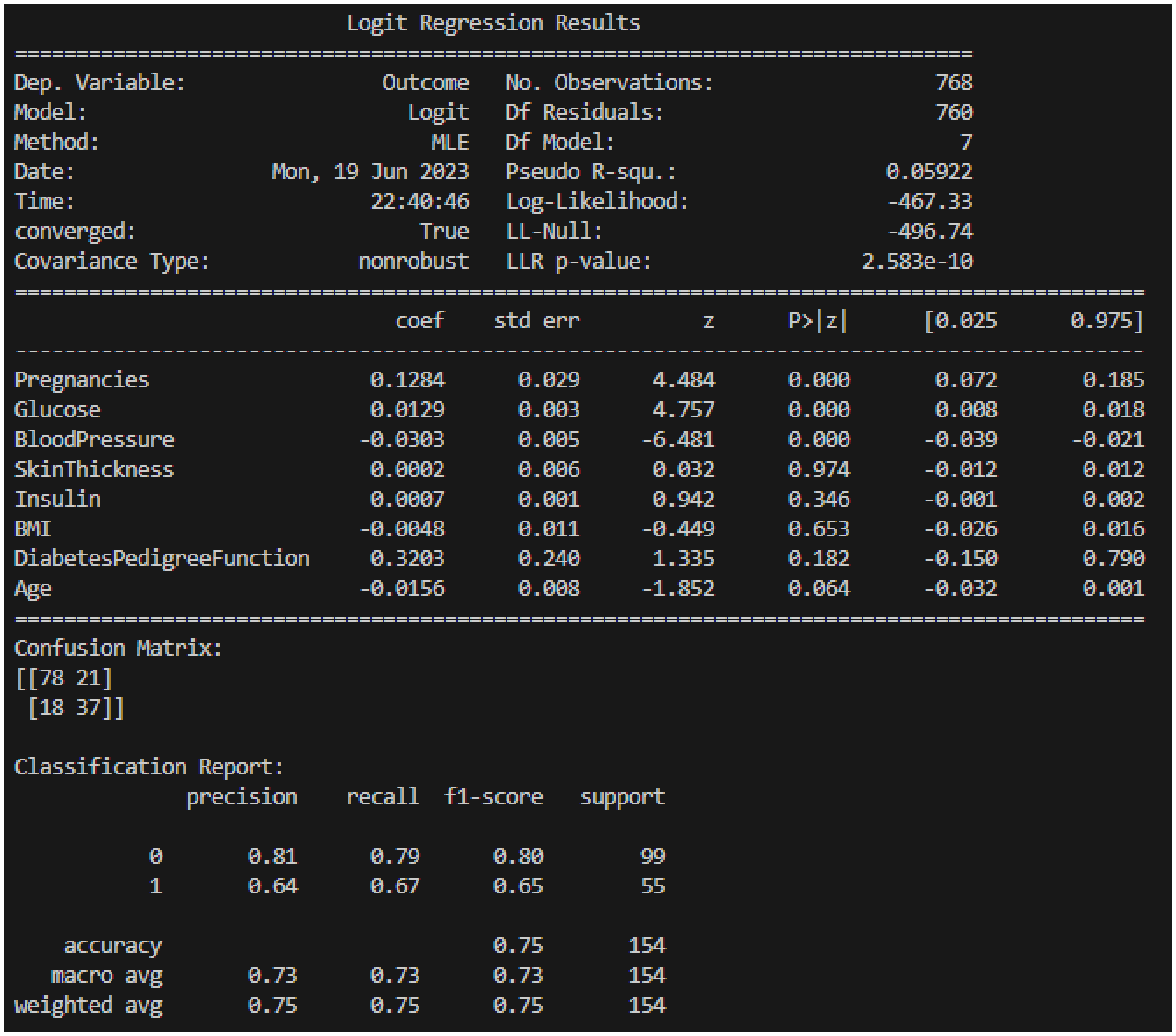
Terminal Output: Multi Linear summary() print

Furthermore, the logistic regression model generation function has a fairly similar structure as it uses both the sklearn and the statsmodel library to create models that uses the training set for both this time. I have this model print out the classes. Coefficients, the summary function food the logistic regression model, a confusion matrix, and a classification report for reference purposes.



```
classes: [0 1]
coefficients: [[ 0.06570398  0.03425924 -0.01386273  0.0035461  -0.00182276  0.10159343
   0.60784931  0.03489913]]
intercept: [-8.92221601]
```

Code Segment: sklearn model terminal output



```
                           Logit Regression Results
==============================================================================
Dep. Variable:                Outcome   No. Observations:                  768
Model:                          Logit   Df Residuals:                      760
Method:                           MLE   Df Model:                            7
Date:                Mon, 19 Jun 2023   Pseudo R-squ.:                 0.05922
Time:                        22:40:46   Log-Likelihood:                -467.33
converged:                       True   LL-Null:                       -496.74
Covariance Type:            nonrobust   LLR p-value:                 2.583e-10
============================================================================================
                               coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Pregnancies                  0.1284      0.029      4.484      0.000       0.072       0.185
Glucose                      0.0129      0.003      4.757      0.000       0.008       0.018
BloodPressure               -0.0303      0.005     -6.481      0.000      -0.039      -0.021
SkinThickness                0.0002      0.006      0.032      0.974      -0.012       0.012
Insulin                      0.0007      0.001      0.942      0.346      -0.001       0.002
BMI                         -0.0048      0.011     -0.449      0.653      -0.026       0.016
DiabetesPedigreeFunction     0.3203      0.240      1.335      0.182      -0.150       0.790
Age                         -0.0156      0.008     -1.852      0.064      -0.032       0.001
============================================================================================
Confusion Matrix:
[[78 21]
 [18 37]]

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        99
           1       0.64      0.67      0.65        55

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.75      0.75      0.75       154
```

Code Segment: statsmodel terminal output

4.
This function generates the multi linear regression model that also can compute the LOOCV error which I have print outputs to see we generate for our error value.The value which we get for our is Multi Linear Regression LOOCV Error: 0.1626687721356742 as read from our terminal output.

```
    # within mlRegression function
    linear_scores = cross_val_score(model, X, y, cv=len(X),
scoring='neg_mean_squared_error')
    linear_loocv_error = -linear_scores.mean()
    print(f"Linear Regression LOOCV Error: {linear_loocv_error}")
```

Code Segment: Multi Linear LOOCV Error

5.

```
    #  mlRegression kfold_error
    linear_scores = cross_val_score(model, X, y, cv=k,
scoring='neg_mean_squared_error')
    linear_kfold_error = -linear_scores.mean()
    print(f"Linear Regression K-Fold CV Error: {linear_kfold_error}")
    if (linear_loocv_error > linear_kfold_error):
        print(f"Linear Regression K-Fold CV Error is better at
{linear_kfold_error}")
    else:
        print(f"Linear Regression LOOCV Error is better at
{linear_kfold_error}")
```

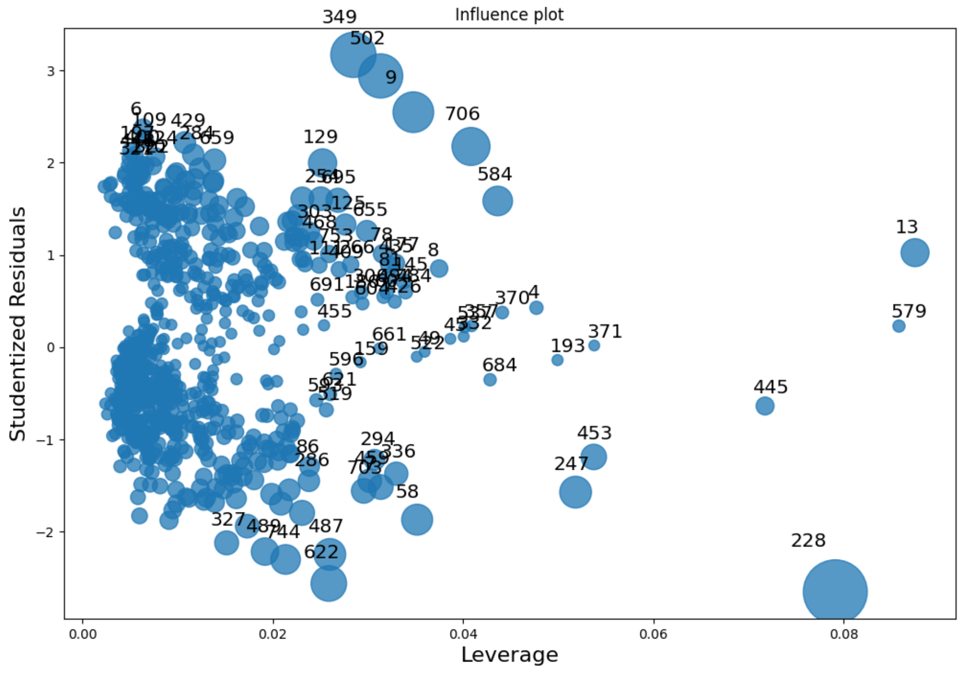
Code Segment: Multi Linear KFOLD Error

```
    # Logistic Regression  kfold error
    logistic_scores = cross_val_score(model, X_train, y_train, cv=k,
scoring='neg_log_loss')
    logistic_kfold_error = -logistic_scores.mean()
    print(f"Logistic Regression K-Fold CV Error: {logistic_kfold_error}")
```

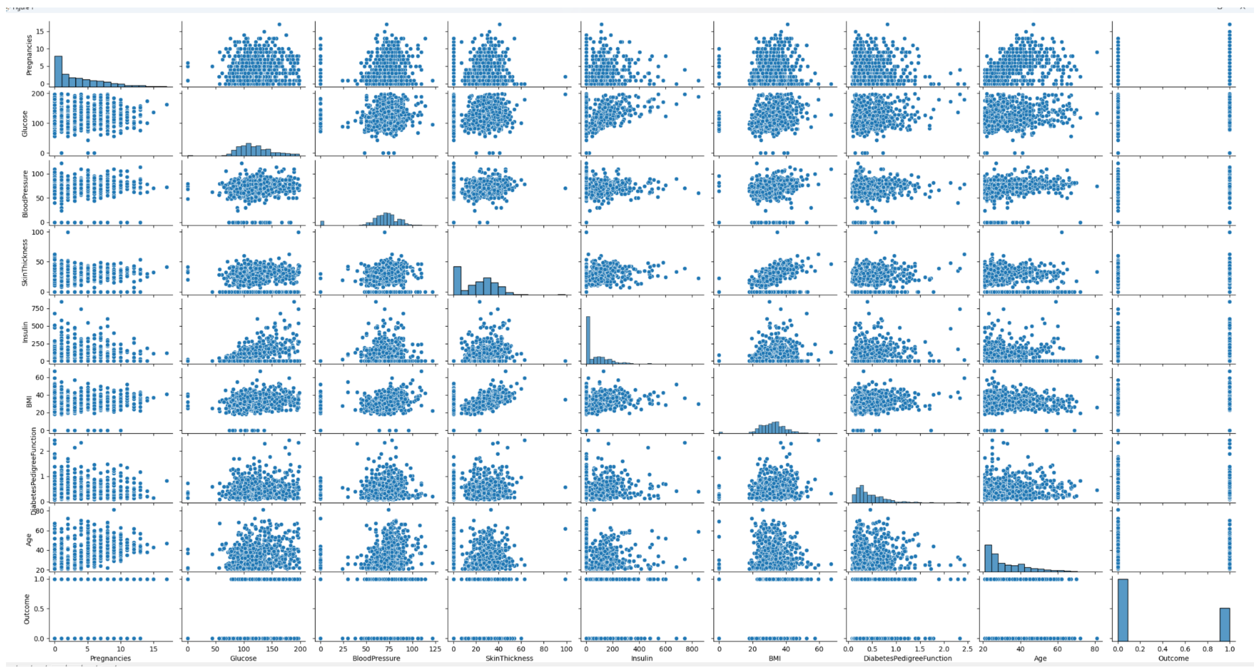
Code Segment: Logistic Regression K-FOLD Error

The Logistic Regression K-Fold CV Error that I received was 0.487, while the Linear Regression K-Fold CV Error is at 0.163
The K-Fold error was lower for both linear regression and logistic regression models.

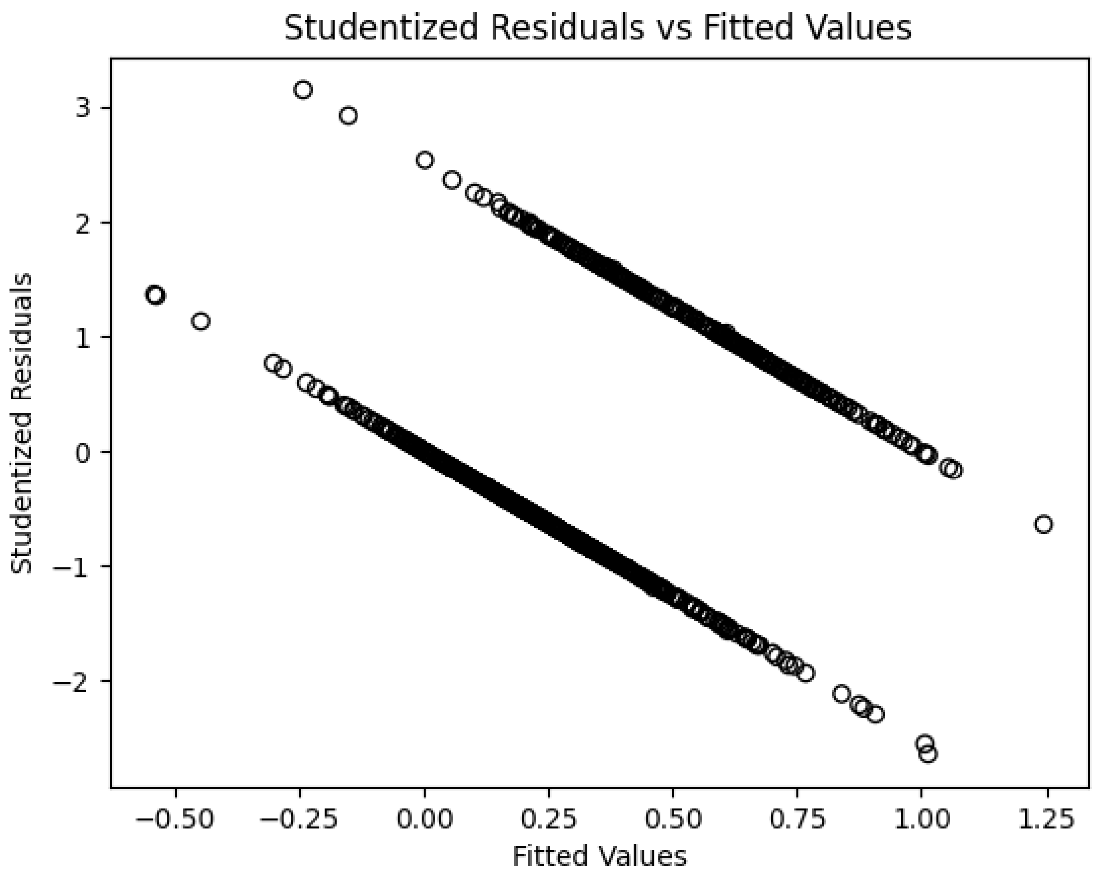
6.



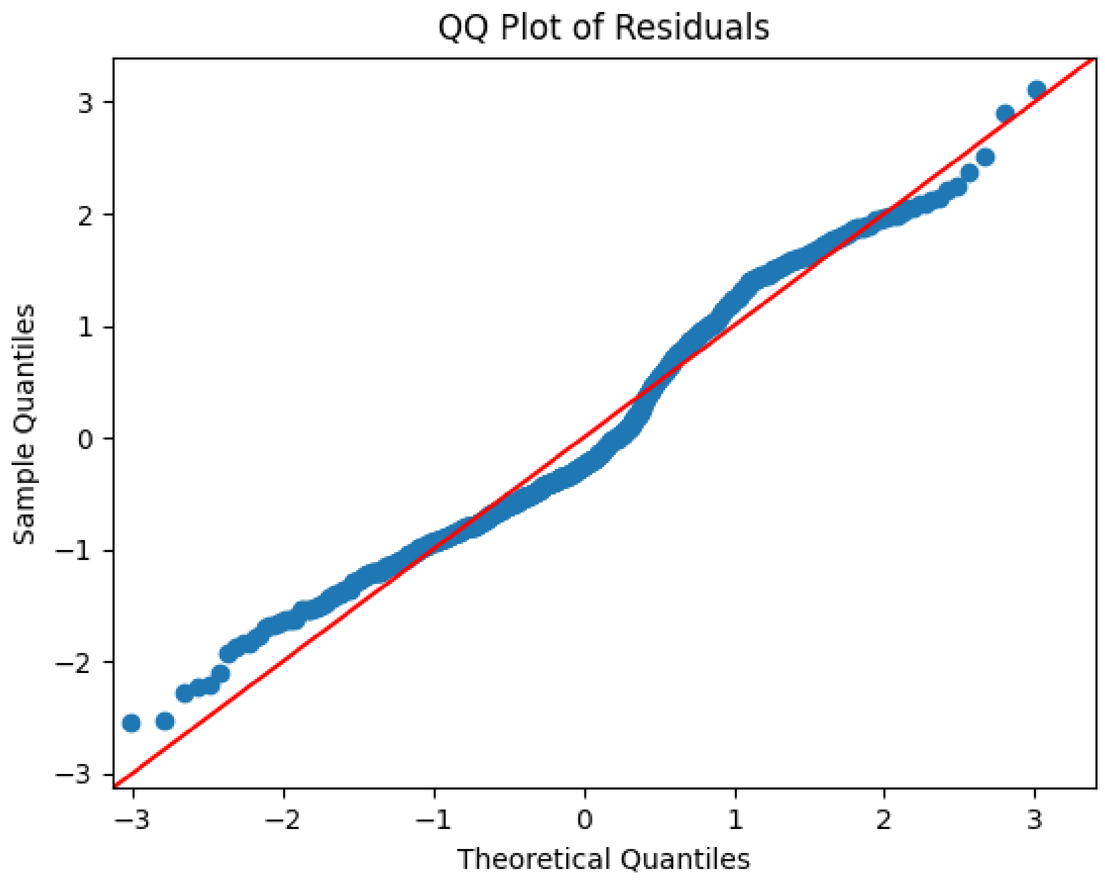
Plot Generation:(a) Influence plot



Code Segment: (b) Pairwise plot



Code Segment: (c) Studentized residual vs predicted response



Code Segment: (d) QQ plot for residuals

Are there any problems with the fit?
Not particularly. I understand that If you see a clear pattern in the residuals plot (like a curve or a funnel shape) in the "Studentized Residual vs Fitted Values" plot and the Q-Q plot, it suggests that the model isn't capturing some aspect of the data, which would be a problem with the fit. On the Q-Q plot all the data plots are not deviating from the line, which implies that all the data is being captured properly with no problems with the fit

Are there any unusually large outliers?
Yes there is a large residual that is particularly weighted and far from the horizontal line at zero. In the bottom right corner of the influence plot.

Are there any observations with unusually high leverage?
228 in particular has an immensely large amount of leverage in comparison to the other data plots which is something to raise an eyebrow about.

Is there a non-linear association between any of the predictors and response?

The pairwise plot allows you to check for non-linear associations. If there is a non-linear association, the scatter plot for that pair of variables will not form a clear straight line.

Is heteroscedasticity present in the model?

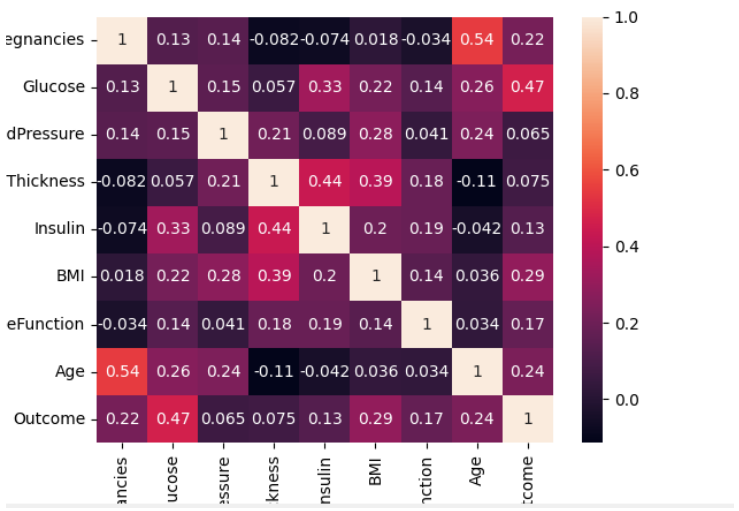
I understand that if the spread of residuals changes along the range of fitted values which looks like a funnel, then heteroscedasticity is present. This doesn't seem to be the case as there are two lines that signify a funnel.

7.

We are asked to look at the correlation matrix of the predictor variables and the corresponding scatter plot matrix to find any evidence of collinearity. I generate such using the code as shown below.

```
sns.heatmap(correlation_matrix, annot=True)
plt.show()
```

Code Segment: Correlation Matrix Generation

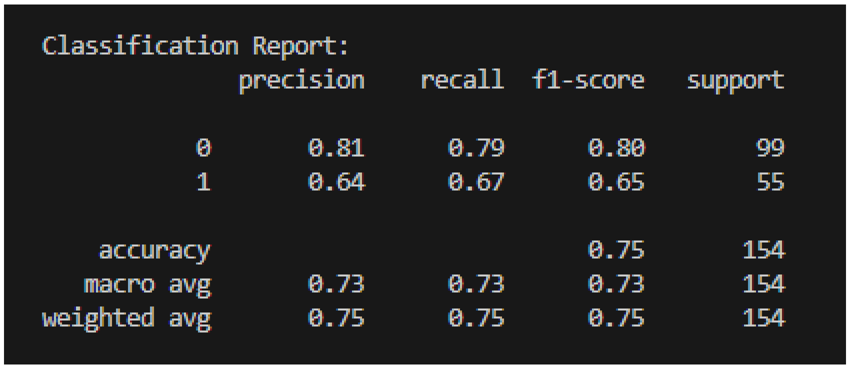


	ancies	ucose	essure	kness	nsulin	BMI	nction	Age	come
egnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
dPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
Thickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
eFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

Terminal Output : Correlation Matrix Plot Generation

We can conclude from this code that there exists the highest collinearity between age and pregnancies as well as outcome and glucose levels.

8.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        99
           1       0.64      0.67      0.65        55

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.75      0.75      0.75       154
```

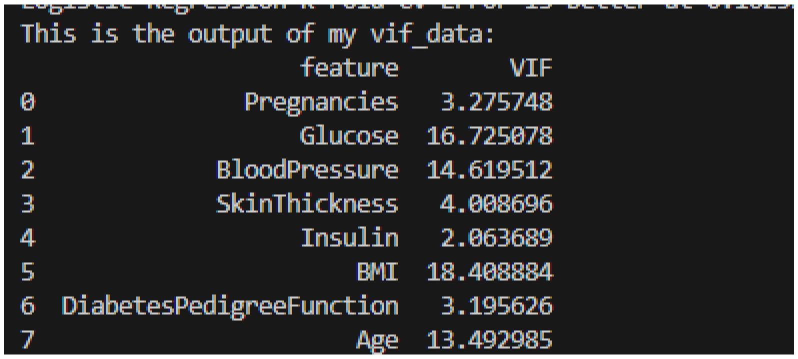
Terminal Output : Classification report for logistic regression model

The classification report does not directly give us anything about colinearity but we can understand that high collinearity can usually lead to improper parameter estimation. We see that precision is correct 81% of the time for class 0, recall is correct 79% of the time for class 0, F1-score is correct 80% of the time for class 0. All the general parameter predictions show that the negative class 0 performs reasonably well and definitely performs better than the class 1 of positives. This did not give me much of an insight on collinearity, so I went in the VIF route to learn more about our model.

```
# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                        for i in range(len(X.columns))]

print(F"This is the output of my vif_data:\n{vif_data}")
```

Code Segment : VIF Parameter dataframe Generation

Any parameter with a value under 5 does not show much of a correlation, while anything above shows a high correlation and if we find a high VIF, we could consider dropping one of the variables, or combining them in some way by using regularization techniques. In my case, Glucose, BloodPressure, BMI, and Age have high VIF values (> 5). This suggests that these predictors may be highly correlated with other predictors in your model. Something to keep in mind for our situation is that a high VIF inflates the variance of the parameter estimate, which makes the significance test less reliable. However, it doesn't bias the parameter estimate itself. As we are primarily interested in prediction rather than interpreting the coefficients, multicollinearity might not be a problem.



```
This is the output of my vif_data:
                    feature        VIF
0               Pregnancies   3.275748
1                   Glucose  16.725078
2             BloodPressure  14.619512
3             SkinThickness   4.008696
4                   Insulin   2.063689
5                       BMI  18.408884
6  DiabetesPedigreeFunction   3.195626
7                       Age  13.492985
```

Code Segment : VIF Parameter Terminal Output

9.

I created an entire other function to create a Principal Component Analysis (PCA) model which is a technique used to emphasize variation and bring out strong patterns in a dataset. It's often used to make data easy to explore and visualize. I first started by performing the actual PCA on the predictors to then use the principal components to build my regression model. Here is how I did it.

```
def pcaGeneration(df):
    # The data
    X = df.drop('Outcome', axis=1)
    y = df['Outcome']

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Perform PCA
    pca = PCA()
    X_pca = pca.fit_transform(X_scaled)
    X_pca_df = pd.DataFrame(data = X_pca)
    model_pca = LinearRegression()

    # Fit the model (use the number of PCs you decided to retain, say 3 for example)
    X_pca_reduced = X_pca_df.iloc[:, :3]
    model_pca.fit(X_pca_reduced, y)

    print('Coefficients:', model_pca.coef_)
```

Code Segment : PCA Generation to get PC Estimates

We can conclude that the coefficients which we obtain as estimates for the pc in the terminal output based on the model_pca.coef function is as shown. Coefficients: [0.12551635 0.0728282 0.08985065].

10.
This is our code generation for Ridge Regression generation

```python
def rrGeneration(df):
    # The data
    X = df.drop('Outcome', axis=1)
    y = df['Outcome']

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Define the alpha values to test
    alphas = np.logspace(-4, 4, 200)  # Feel free to adjust this

    ridge_cv = RidgeCV(alphas=alphas, store_cv_values=True)
    ridge_cv.fit(X_scaled, y)

    best_alpha = ridge_cv.alpha_
    print('Best alpha:', best_alpha)

    ridge = RidgeCV(alphas=[best_alpha])
    ridge.fit(X_scaled, y)
    print('Ridge coefficients:', ridge.coef_)
```

Code Segment : Ridge Regression Model Generation

We first define the alpha values then we create a ridge model based on the cv values which we have saved as well as the alpha values we had generated. The best alpha is automatically chosen for us which is found to be the value of 38.724. Then we find the regression coefficients by using this value of k and I have listed them down below.

```
Ridge coefficients: [ 0.0658565   0.17889324 -0.04048852  0.00184063 -0.0157017   0.0997257
  0.04722364  0.03300364]
```

Conclusion:
There were no particular problems with the data collected for the experiment and model assumptions. I was able to circumvent most problems by referencing the internet for help as well as lecture references. I did detect large amounts of collinearity between the predictors as almost all of them showed large signs of it based on VIF and the plots. I chose to retain 3 PCSs. The model I would recommend to describe the relationship between the response and predictor variables is the logistic regression model!

Final Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import numpy as np
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.graphics.regressionplots import influence_plot
from statsmodels.stats.outliers_influence import OLSInfluence
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import LogisticRegression,LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RidgeCV

k = 5 #for k means

# Load the diabetes dataset
df = pd.read_csv('diabetes.csv')
print(df)

# Function to visualize the distribution of each feature
def visualize_features(df):
    for column in df.columns:
        plt.figure()
        sns.histplot(df[column], bins=20, kde=True)
        plt.title(column)
        plt.show()

# Function to perform multiple linear regression and evaluate the model
def mlRegression(df,k):
    columns_selected = "+".join(df.columns.difference(["Outcome"]))
    my_formula = "Outcome~" + columns_selected

    X = df.drop('Outcome', axis=1)
    y = df['Outcome']
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Create a linear regression model
    model = LinearRegression()
    lm_fit = smf.ols(my_formula, data=df).fit()
    print(lm_fit.summary())
    lm_fit.resid.describe()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print('Mean Squared Error:', mse)
    print('R-squared Score:', r2)
    # within mlRegression function
    linear_scores = cross_val_score(model, X, y, cv=len(X),
scoring='neg_mean_squared_error')
    linear_loocv_error = -linear_scores.mean()
    print(f"Linear Regression LOOCV Error: {linear_loocv_error}")

    #  mlRegression kfold_error
    linear_scores = cross_val_score(model, X, y, cv=k,
scoring='neg_mean_squared_error')
    linear_kfold_error = -linear_scores.mean()
    print(f"Linear Regression K-Fold CV Error: {linear_kfold_error}")
    if (linear_loocv_error > linear_kfold_error):
        print(f"Linear Regression K-Fold CV Error is better at
{linear_kfold_error}")
    else:
        print(f"Linear Regression LOOCV Error is better at
{linear_kfold_error}")

    #influence plot
    fig, ax = plt.subplots(figsize=(12,8))
    influence_plot(lm_fit, ax=ax)
    plt.title("Influence plot")
    plt.show()
    sns.pairplot(df)
    plt.show()

    #Studentized Residual vs Predicted Response:
    student_resid = OLSInfluence(lm_fit).resid_studentized_internal
    pred_vals = lm_fit.predict(X)
    plt.scatter(pred_vals, student_resid, edgecolors='k',
facecolors='none')
    plt.ylabel('Studentized Residuals')
    plt.xlabel('Fitted Values')
    plt.title('Studentized Residuals vs Fitted Values')
    plt.show()

    #qqplot for residuals
    res = lm_fit.resid # residuals
    fig = sm.qqplot(res, fit=True, line='45')
    plt.title('QQ Plot of Residuals')
    plt.show()

    # VIF dataframe
    vif_data = pd.DataFrame()
    vif_data["feature"] = X.columns

    # calculating VIF for each feature
    vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                            for i in range(len(X.columns))]

    print(F"This is the output of my vif_data:\n{vif_data}")


# Function to perform logistic regression and evaluate the model
def logisticRegression(df,k):
    # Split the dataset into features (X) and target (y)
    X = df.drop('Outcome', axis=1)
    y = df['Outcome']
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Create a logistic regression model
    model = LogisticRegression()
    smLogit = sm.Logit(y,X)
    model.fit(X_train, y_train)

    result = smLogit.fit()

    print('classes:', model.classes_)
    print('coefficients:', model.coef_)
    print('intercept:', model.intercept_)
    print(result.summary())

    # Predict the target values for the test data
    y_pred = model.predict(X_test)

    # Evaluate the model
    print('Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))

    print('\nClassification Report:')
    print(classification_report(y_test, y_pred))
    print(model)
    correlation_matrix = df.corr()

    sns.heatmap(correlation_matrix, annot=True)
    plt.show()

    # # within logistic_regression function
    # logistic_scores = cross_val_score(model, X, y, cv=len(X),
scoring='neg_log_loss')
    # logistic_loocv_error = -logistic_scores.mean()
    # print(f"Logistic Regression LOOCV Error: {logistic_loocv_error}")


    # Logistic Regression  kfolderror
    logistic_scores = cross_val_score(model, X_train, y_train, cv=k,
scoring='neg_log_loss')
```

```
    logistic_kfold_error = -logistic_scores.mean()
    print(f"Logistic Regression K-Fold CV Error: {logistic_kfold_error}")

    # if (logistic_loocv_error > logistic_kfold_error):
    #     print(f"Logistic Regression K-Fold CV Error is better at 
{logistic_kfold_error}")
    # else:
    #     print(f"Logistic Regression LOOCV Error is better at 
{logistic_kfold_error}")

def pcaGeneration(df):
    # The data
    X = df.drop('Outcome', axis=1)
    y = df['Outcome']

    # Standardize the features (important for PCA)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Perform PCA
    pca = PCA()
    X_pca = pca.fit_transform(X_scaled)
    X_pca_df = pd.DataFrame(data = X_pca)
    model_pca = LinearRegression()

    X_pca_reduced = X_pca_df.iloc[:, :3]  # adjust this based on the 
number of PCs you want to retain
    model_pca.fit(X_pca_reduced, y)

    # Print the coefficients
    print('Coefficients:', model_pca.coef_)

def rrGeneration(df):
    # The data
    X = df.drop('Outcome', axis=1)
    y = df['Outcome']

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    alphas = np.logspace(-4, 4, 200)  # Feel free to adjust this
    ridge_cv = RidgeCV(alphas=alphas, store_cv_values=True)
    ridge_cv.fit(X_scaled, y)

    best_alpha = ridge_cv.alpha_
    print('Best alpha:', best_alpha)

    ridge = RidgeCV(alphas=[best_alpha])
    ridge.fit(X_scaled, y)
    print('Ridge coefficients:', ridge.coef_)

# Visualize the distribution of each feature
# Perform logistic regression and evaluate the model
# visualize_features(df)
# mlRegression(df,k)
# logisticRegression(df,k)
# pcaGeneration(df)
rrGeneration(df)
```