Homework 4

Jordan Small - Z23465928

April 23, 2024

CAP4773 – Data Science and Analytics

Spring 2024

Dr. Juan Yepes

Florida Atlantic University

The Diabetes Dataset, selected from Kaggle, maintains the objective of diagnostically predicting a patient's likelihood of developing diabetes. After placing several constraints, there were 768 instances selected from a larger dataset. All patients (instances), in particular, are females-- at least 21 years old-- of Pima Indian heritage. This dataset uses eight numerical predictors to compare with one binary outcome attribute. These predictor attributes are all based on certain diagnostic measurements and include the following: Pregnancies (to express the number of pregnancies), Glucose (to express the glucose level in blood), BloodPressure (to express the blood pressure measurement), SkinThickness (to express the thickness of the skin), Insulin (to express the insulin level in blood), BMI (to express the body mass index), DiabetesPedigreeFunction (to express the Diabetes percentage), and Age (to express the age). The binary attribute, Output, expresses the result (whether the patient has diabetes) represented by a 1 for yes and a 0 for no. Throughout the analysis, the task is to predict the outcome based on the other features.

In order to evaluate the data, several steps were put in place for data preparation. All required libraries were imported to a Jupyter Notebook where the dataset was loaded. In doing so, visualizations were made to show the aforementioned data features and table/index info. In order to groom the data (handling missing data in particular), N/A values were dropped from the dataset. This will allow for full functionality for the tools in these libraries and provide clearer calculations and analyses when doing so.

The next step was exploratory data analysis. The goal in this step was to analyze basic trend statistics by generating a scatter plot matrix and analyzing multicollinearity by using a Variance Inflation Factor (VIF). The scatter plot matrix made some relationships amongst attributes very clear in a visual way. For example, there are observable linear relationships between some of the following variables: glucose and insulin, skin thickness and age and BMI and insulin (to name a few). Analyzing the calculated VIF's: BMI, blood pressure, and glucose levels all have a relatively high VIF, indicating that there is multicollinearity with other predictors; age has a moderate VIF, indicating that there might be multicollinearity amongst

some predictors; and attributes such as insulin, skin thickness, and pregnancies all have a relatively low VIF, indicating that there is no clear multicollinearity amongst other features.

After determining attributes that display clear relationships, the next phase was to setup the data to prepare for drafting models. At this point, it was very important to prepare the Python environment and make sure that all necessary libraries were imported. Before and during the testing of individual models, it was also necessary to set up feature (X parameter) and target (y parameter) data. Once determined, the dataset would need to be split into training and testing sets for validation. Some models will require manipulation of these data subsets. When modeling, predictor variables such as pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, age, and the diabetes pedigree function were selected as the X (independent features) and the outcome was selected as the Y (target variable).

The first model to test was Multiple Linear Regression. The data was standardized and a linear regression model was trained using imported libraries. The Multiple Linear Regression model aimed to predict outcomes from selected health attributes, resulting in a Mean Squared Error (MSE) of approximately 0.171, indicating relatively close predictions to actual values. However, there is residual variance unaccounted for. An R-squared error of 0.255 indicates that about 25.5% of the variance in outcome is explained by the model, suggesting a slight fit. The chosen features have some overall influence on the outcome of whether the patient has diabetes, capturing a trend in the data, but there is room for improvement—perhaps through feature exclusion or utilizing more complex modeling methodologies.

Next up was evaluating linear regression using regularization. To do so, the data (including the features and the target) were standardized then a Lasso Regression Model was trained using an alpha value of 0.05. This resulted in a model where some coefficients were reduced to zero, indicating that the corresponding features were deemed less important by the regularization process. Specifically, the

model retained ‘Pregnancies’, ‘Glucose’, ‘BMI’, ‘DiabetesPedigreeFunction’, and ‘Age’ as influential factors with their coefficients being 0.04, 0.33, 0.18, 0.03, and 0.12 respectively. Other features, such as ‘SkinThickness’ and ‘Insulin’ completely eliminated from the model (with a coefficient of 0) and ‘BloodPressure’ was reduced to a minimum value of –0.002. The model’s intercept is 1.54e-16, serving as a baseline prediction when all features are at their mean values. The Mean Squared Error (MSE) of this model is roughly 0.173, which suggests that the model’s predictions are reasonably close to the actual outcome. This MSE was barely an improvement over that of the Multiple Linear Regression model. The R-square value is approximately 0.25, indicating that nearly 25% of the variability in the response variable is explained by the model. The overall performance of this model is comparable to the non-regularized Multiple Linear Regression model, indicating that Lasso’s feature selection did not significantly change the predictive capability in this case.

Polynomial regression models exhibited a significant difference in performance. Both the 4^{th} and 5^{th} degree models suffered from overfitting, evidenced by high MSE’s of 244.41 and 2312.83, respectively. The 3^{rd} degree model demonstrated a much better fit and generalization, achieving a relatively low MSE of 1.36. Compared to the previous models which had lower MSE’s (and therefore higher predictive accuracies), Multiple Linear Regression and Lasso models outperformed in this context. Lasso's regularization effectively countered overfitting, making it a more reliable choice than overly complex polynomial models.

The Decision Tree Regressor, set to a maximum depth of 3, yielded a Mean Squared Erorr (MSE) of approximately 0.176, indicating reasonably similar predictions to the previous model. The R-squared value of about 0.24 is in-line with the other models as well, suggesting that the model explains about a quarter of the variance in the target variable—which is relatively low predictive performance. This model still provides valuable insights, but is about as accurate as previous models and thus could benefit from further manipulation or integration with ensemble methods to improve prediction accuracy.

At this point, all that was left was to evaluate the models in preparation for the deployment phase. In doing so, models were evaluated using cross-validation. The Lasso Regression Model using cross-validation yielded a chosen alpha of approximately 0.0077. Its coefficients were all very low (with values as little as 3.63e-04 and as high as 2.37e-01) with an intercept at virtually zero (1.8e-16). The MSE and R-squared values were in-line with the other models at 0.17 and 0.26 respectively. Since these models all performed comparably in terms of MSE and R-squared, Lasso Regression (through its regularization) effectively reduced the complexity of the model by reducing some coefficients to zero, potentially improving the model interpretability and preventing overfitting.

That said, the selected model was the Lasso Regression Model. In the deployment phase, predictions were made using new data and feeding it to the model. A new dataset was generated, selecting five random points for each variable. Randomness was insured by using uniform random distribution, from one of the imported libraries. The results of the model were printed and values ranged from 0.01 to 0.52, where numbers closer to 1 indicated an outcome where the patients were more likely to have diabetes.

Homework4

April 23, 2024

1 Homework 4 - Final Project

1.1 Step 1: Problem Understanding

[75]:
```
# Based on health attributes in order to diagnostically predict diabetes in a␣
 ↪patient

# 1.1 Dataset selection:

# Name: Diabetes Dataset

# Source: https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset

# Description: The objective of the dataset is to diagnostically predict␣
 ↪whether a patient has diabetes, based on certain diagnostic measurements␣
 ↪included in the dataset

# Number of instance: 768

# Number of attributes: 9- 8 numerical predictors and a binary output
# Attribute information:
# 1. Pregnancies: To express the Number of pregnancies
# 2. Glucose: To express the Glucose level in blood
# 3. BloodPressure: To express the Blood pressure measurement
# 4. SkinThickness: To express the thickness of the skin
# 5. Insulin: To express the Insulin level in blood
# 6. BMI: To express the Body mass index
# 7. DiabetesPedigreeFunction: To express the Diabetes percentage
# 8. Age: To express the age
# 9. Outcome: To express the final result 1 is Yes and 0 is

# 1.2 Defining a prediction task
# TASK: Predict one of the features (outcome) based on the others.
```

1.2 Step 2: Data Preparation

[34]:
```
# 2.1 Import required libraries

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
import statsmodels.api as sm
from statsmodels.discrete.discrete_model import Logit

from sklearn import datasets
from sklearn.linear_model import LinearRegression, Lasso, LassoCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import cross_val_score, train_test_split, KFold
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
 ↪mean_squared_error, r2_score
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor,␣
 ↪plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
from sklearn.pipeline import Pipeline

# 2.2 Load the dataset
data = pd.read_csv('diabetes.csv')

# Show the first few rows of the DataFrame
data.head()
```

[34]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0

```
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[18]:
```
# Show feature types
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[20]:
```
# 2.2 Handling missing data

data = data.dropna()
```

1.3 Step 3: Exploratory Data Analysis

[23]:
```
# 3.1 Analyzing basic trend statistics, generating box plots and scatter plots,␣
 ↪among others.

# 3.1.1 Trend statistics
data.describe()
```

[23]:

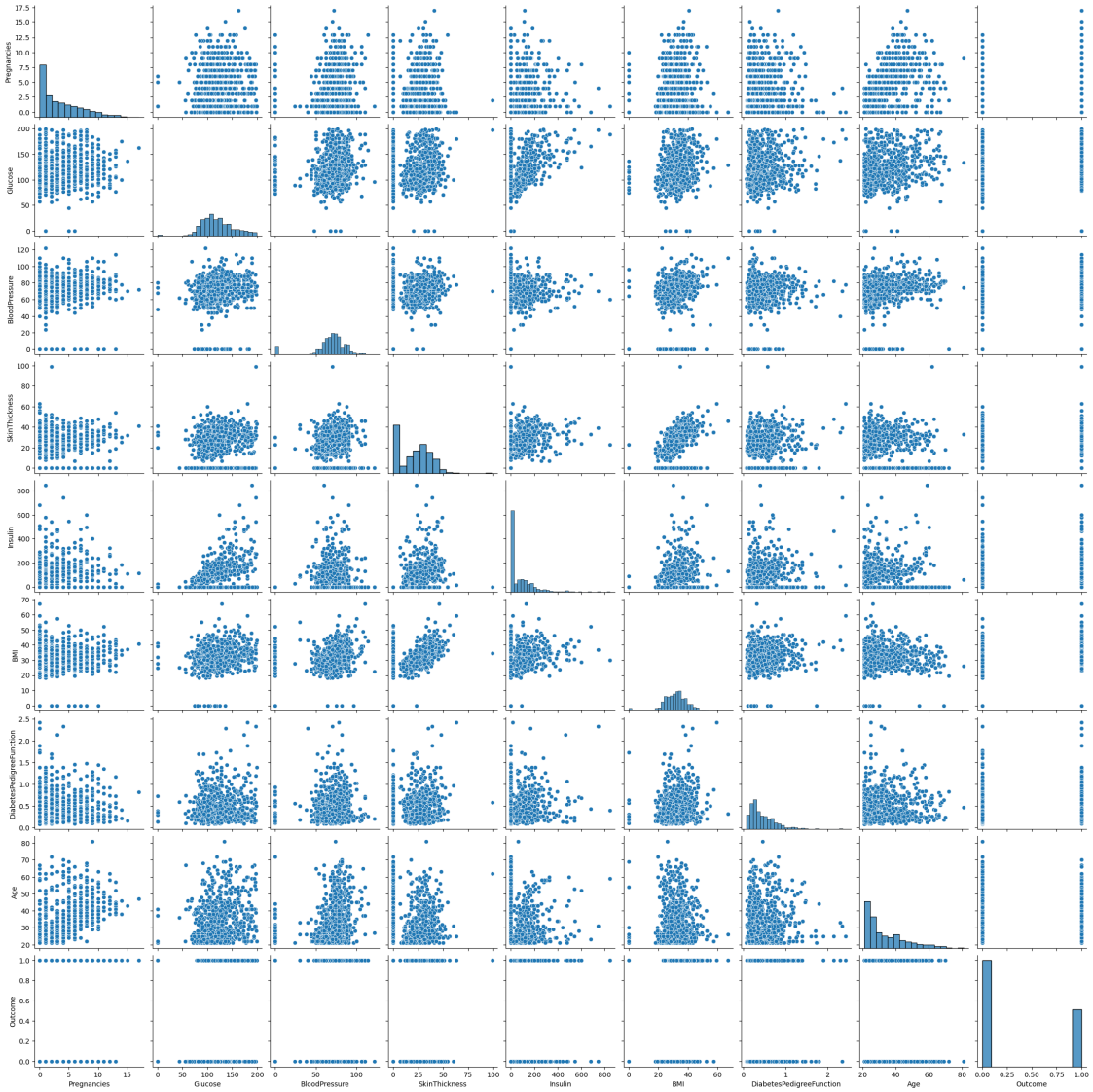
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	
std	3.369578	31.972618	19.355807	15.952218	115.244002	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	

	BMI	DiabetesPedigreeFunction	Age	Outcome

```
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[26]:
```
# Create a scatter plot matrix
sns.pairplot(data)
plt.show()
```

```
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```



[38]:
```
#3.2. Conducting correlation and multicollinearity analysis

#3.2.1. Variance Inflation Factor
def calculate_vif(data):
    vif_data = pd.DataFrame()
    vif_data["Feature"] = data.columns
    vif_data["VIF"] = [variance_inflation_factor(data.values, i)
                       for i in range(data.shape[1])]
    return vif_data

print(calculate_vif(data))
```

```
                    Feature         VIF
```

```
0                Pregnancies   3.364416
1                    Glucose  18.030209
2              BloodPressure  15.313159
3              SkinThickness   4.008709
4                    Insulin   2.063940
5                        BMI  18.515112
6  DiabetesPedigreeFunction   3.213450
7                        Age  13.500531
8                    Outcome   1.948398
```

1.4 Step 4: Setup Phase

[71]:
```
# 4.1. Preparing the Python environment, importing libraries. - Done in 2.1
# 4.2. Setting up your X (features) and Y (target) data. (Done before each␣
 ↪model)
# 4.3. Split the dataset into training and testing sets for validation (Done␣
 ↪before each model)
```

1.5 Step 5: Modeling Phase

[44]:
```
# Select predictor variables (x) and target variable (y)

X = data[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
 ↪'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
X = sm.add_constant(X) # Adds a constant column to input features to account␣
 ↪for the intercept

y = data['Outcome'] # Use 'Outcome' ass the target variable

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

[80]:
```
# 5.1.        Multiple Linear Regression

# 5.1.1 Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

# 5.1.2 Train a linear regression model
linear_reg = LinearRegression()
linear_reg.fit(X_scaled, y_train)

# 5.1.3 Standardizing the test set
X_test_scaled = scaler.transform(X_test)

# 5.1.4 Predicting outcome using the trained model
```

```
y_pred = linear_reg.predict(X_test_scaled)

# 5.1.7 Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# 5.1.8 Display the model statistics
print(f"Coefficients: {linear_reg.coef_}")
print(f"Intercept: {linear_reg.intercept_}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```

```
Coefficients: [ 0.03465559  0.1803234  -0.04219339  0.00820563 -0.03230381
0.11631364
  0.03744793  0.07425473]
Intercept: 0.34690553745928343
Mean Squared Error (MSE): 0.17104527280850096
R-squared (R2): 0.2550028117674178
```

[82]:
```
# 5.2 Linear Regression using Regularization

# 5.2.1 Standardizing the features
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

# 5.2.2 Standardizing the target variable
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).
 ↪flatten()  # Reshape for a single feature

# 5.2.3 Training a Lasso regression model using some alpha
lasso_reg = Lasso(alpha=0.05)
lasso_reg.fit(X_train_scaled, y_train_scaled)

# 5.2.4 Predicting 'outcome' using the trained Lasso regression model
y_pred_scaled = lasso_reg.predict(X_test_scaled)

# 5.2.5 Reversing the scaling of predictions to original scale
y_pred_lasso = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1)).
 ↪flatten()

# 5.2.6 Evaluating the model's performance
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

# 5.2.7 Displaying the model statistics
```

```
print("Lasso Regression Model")
print(f"Coefficients: {lasso_reg.coef_}")
print(f"Intercept: {lasso_reg.intercept_}")
print(f"Mean Squared Error (MSE): {mse_lasso}")
print(f"R-squared (R2): {r2_lasso}")
```

```
Lasso Regression Model
Coefficients: [ 0.04152994  0.32609713 -0.00193766 -0.         -0.
0.18395707
  0.03261341  0.11838998]
Intercept: 1.5408749945421284e-16
Mean Squared Error (MSE): 0.17261677225081198
R-squared (R2): 0.24815805864090756
```

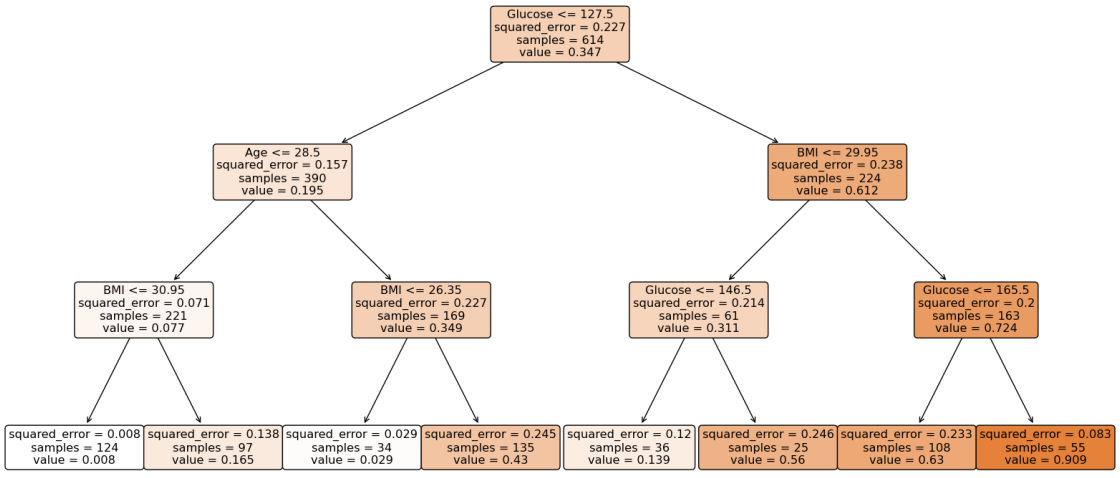
[50]:
```
#5.3 Polynomial Regression

# 5.3.1 Using a pipeline for 3rd-degree polynomial regression
poly3_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # First, scale the features
    ('poly', PolynomialFeatures(degree=3)),  # Then, generate polynomial␣
 ↪features
    ('linear', LinearRegression())  # Finally, apply linear regression
])

# 5.3.2  Fit the pipeline on the training data
poly3_pipeline.fit(X_train, y_train)

# 5.3.3 Predict on the test data
y_pred_poly3 = poly3_pipeline.predict(X_test)

# 5.3.4 Calculate MSE
mse_poly3 = mean_squared_error(y_test, y_pred_poly3)

# 5.3.5 Print the MSE for the 3rd-degree polynomial model
print(f"Mean Squared Error for 3rd-degree Polynomial: {mse_poly3:.2f}")

# 5.3.6 Using a pipeline for 4th-degree polynomial regression
poly4_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # First, scale the features
    ('poly', PolynomialFeatures(degree=4)),  # Then, generate polynomial␣
 ↪features
    ('linear', LinearRegression())  # Finally, apply linear regression
])

# 5.3.7 Fit the pipeline on the training data
poly4_pipeline.fit(X_train, y_train)
```

```
# 5.3.8 Predict on the test data
y_pred_poly4 = poly4_pipeline.predict(X_test)

# 5.3.9 Calculate MSE
mse_poly4 = mean_squared_error(y_test, y_pred_poly4)

# 5.3.10 Print the MSE for the 4th-degree polynomial model
print(f"Mean Squared Error for 4th-degree Polynomial: {mse_poly4:.2f}")

# 5.3.11 Using a pipeline for 5th-degree polynomial regression
poly5_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # First, scale the features
    ('poly', PolynomialFeatures(degree=5)),  # Then, generate polynomial␣
 ↪features
    ('linear', LinearRegression())  # Finally, apply linear regression
])

# 5.3.12 Fit the pipeline on the training data
poly5_pipeline.fit(X_train, y_train)

# 5.3.13 Predict on the test data
y_pred_poly5 = poly5_pipeline.predict(X_test)

# 5.3.14 Calculate MSE
mse_poly5 = mean_squared_error(y_test, y_pred_poly5)

# 5.3.15 Print the MSE for the 5th-degree polynomial model
print(f"Mean Squared Error for 5th-degree Polynomial: {mse_poly5:.2f}")
```

```
Mean Squared Error for 3rd-degree Polynomial: 1.36
Mean Squared Error for 4th-degree Polynomial: 244.41
Mean Squared Error for 5th-degree Polynomial: 2312.83
```

[62]:
```
#5.4.        Decision Tree for regression

# Select predictor variables (x) and target variable (y)

X = data[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
 ↪'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]

y = data['Outcome'] # Use 'Outcome' ass the target variable

# Split the dataset into training and testing sets for validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Creating a Decision Tree Regressor
```

```
regressor = DecisionTreeRegressor(max_depth=3, random_state=42)

# Training the model
regressor.fit(X_train, y_train)
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
 ↪'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']

# Plotting the Decision Tree
plt.figure(figsize=(20,10))
plot_tree(regressor, filled=True, feature_names=feature_names, rounded=True)
plt.show()

# Predicting the outcome in the test set
y_pred = regressor.predict(X_test)

# Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
# R-squared Value
r2 = r2_score(y_test, y_pred)

# Displaying the statistics
print(f"Decision Tree Regressor")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```




```
Decision Tree Regressor
Mean Squared Error (MSE): 0.17559792403858912
R-squared (R2): 0.23517348640970048
```

[78]:
```
#6 Evaluating Model
# 6.1.1 Linear Regression using Lasso Regularization with cross validation

# Standardizing the features
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

# Standardizing the target variable
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).
 ↪flatten()  # Reshape for a single feature

# Creating and training a Lasso regression model with cross-validation to find␣
 ↪the best alpha
lasso_cv = LassoCV(alphas=np.logspace(-6, 1, 10), cv=5, random_state=42)
lasso_cv.fit(X_train_scaled, y_train_scaled)

# Predicting 'outcome' using the trained Lasso regression model
y_pred_scaled = lasso_cv.predict(X_test_scaled)

# Reversing the scaling of predictions to the original scale
y_pred_lasso = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1)).
 ↪flatten()

# Evaluating the model's performance
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

# Displaying the model statistics
print("Lasso Regression Model with Cross-Validation")
print(f"Chosen Alpha: {lasso_cv.alpha_}")
print(f"Coefficients: {lasso_cv.coef_}")
print(f"Intercept: {lasso_cv.intercept_}")
print(f"Mean Squared Error (MSE): {mse_lasso}")
print(f"R-squared (R2): {r2_lasso}")
```

```
Lasso Regression Model with Cross-Validation
Chosen Alpha: 0.00774263682681127
Coefficients: [ 6.85588200e-02  3.67108394e-01 -7.36317377e-02  3.63480898e-04
 -4.86547605e-02  2.38977802e-01  7.25919946e-02  1.48648386e-01]
Intercept: 1.8443222184796637e-16
Mean Squared Error (MSE): 0.17030668378371006
R-squared (R2): 0.25821977729761825
```

1.6 Step 7: Deployment Phase

[69]:
```
# 7.1. Make predictions and classifications with new data
# Lasso Regression Model

#7.1.1. Manually building df_new based on the sampled data
import pandas as pd
import statsmodels.api as sm

# Select 5 random points for each variable
random_points = pd.DataFrame({
    'Pregnancies': np.random.uniform(data['Pregnancies'].min(),␣
 ↪data['Pregnancies'].max(), size=5),
    'Glucose': np.random.uniform(data['Glucose'].min(), data['Glucose'].max(),␣
 ↪size=5),
    'BloodPressure': np.random.uniform(data['BloodPressure'].min(),␣
 ↪data['BloodPressure'].max(), size=5),
    'SkinThickness': np.random.uniform(data['SkinThickness'].min(),␣
 ↪data['SkinThickness'].max(), size=5),
    'Insulin': np.random.uniform(data['Insulin'].min(), data['Insulin'].max(),␣
 ↪size=5),
    'BMI': np.random.uniform(data['BMI'].min(), data['BMI'].max(), size=5),
    'DiabetesPedigreeFunction': np.random.
 ↪uniform(data['DiabetesPedigreeFunction'].min(),␣
 ↪data['DiabetesPedigreeFunction'].max(), size=5),
    'Age': np.random.uniform(data['Age'].min(), data['Age'].max(), size=5),
    'Outcome': np.random.uniform(data['Outcome'].min(), data['Outcome'].max(),␣
 ↪size=5),
})


X_new = random_points[['Pregnancies', 'Glucose', 'BloodPressure',␣
 ↪'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
X_new = sm.add_constant(X_new)

# 7.1.2. Lasso Regression Model - TASK 3
# Ensure to scale new data as per the trained model's scaling
from sklearn.preprocessing import StandardScaler

scaler_X = StandardScaler()  # Assuming StandardScaler was used during training
X_new_scaled = scaler_X.fit_transform(X_new)

# Assuming lasso_reg is the trained Lasso Regression model
y_pred_lasso = lasso_reg.predict(X_new_scaled)

# Assuming scaler_y is the scaler used for the target variable during training
```

```
y_pred_lasso_actual = scaler_y.inverse_transform(y_pred_lasso.reshape(-1, 1)).
 ↪flatten()  # If the output was scaled

# 7.1.3. Printing results
#Header
print(f"{'':<6}{'Lasso Regression':<15}")
print(f"{'Index':<6} {'Task 3':<15}")

# Rows of the table
for index, lasso in enumerate(y_pred_lasso_actual):
    print(f"{index:<6}{lasso:<15.3f}")
```

```
      Lasso Regression
Index  Task 3
0     0.502
1     0.521
2     0.335
3     0.365
4     0.010
```

[ ]: