In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, r
import seaborn as sns
```

In [4]:
```
data = pd.read_csv('C:/Users/Adamin/OneDrive/Desktop/diabetes.csv')
```

In [5]:
```
print(data.head())
```

```
   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

   Pedigree  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
```

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
Pedigree         0
Age              0
Outcome          0
dtype: int64
```

In [7]:
```
# Replace zeros with mean for selected columns
cols_to_replace = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
for column in cols_to_replace:
    data[column].replace(0, np.nan, inplace=True)
    data[column].fillna(round(data[column].mean(skipna=True)), inplace=True)
```

```
C:\Users\Adamin\AppData\Local\Temp\ipykernel_10756\167787148.py:4: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[column].replace(0, np.nan, inplace=True)
C:\Users\Adamin\AppData\Local\Temp\ipykernel_10756\167787148.py:5: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[column].fillna(round(data[column].mean(skipna=True)), inplace=True)
```

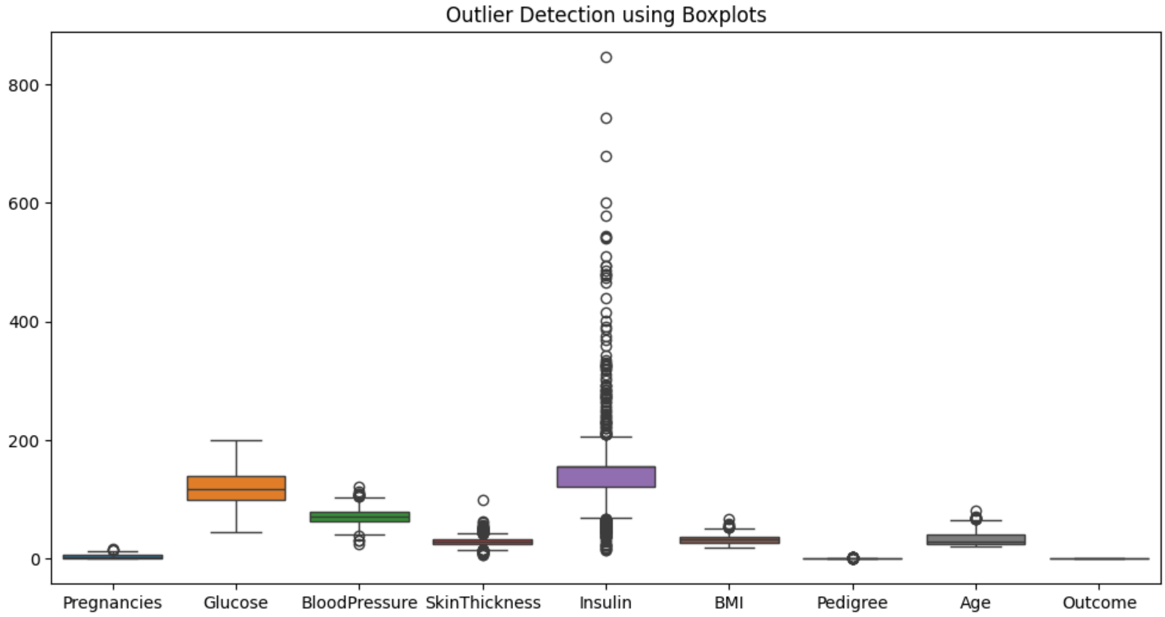
In [8]:

```python
# Features and target
X = data.iloc[:, :8]   # first 8 columns are features
Y = data['Outcome']    # target column
# Split data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_
import matplotlib.pyplot as plt

# Visualize outliers using boxplots
plt.figure(figsize=(12, 6))
sns.boxplot(data)
plt.title("Outlier Detection using Boxplots")
plt.show()

# Identify outliers using IQR
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

# Display count of outliers per column
outliers = ((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).sum()
print("\nNumber of Outliers per Feature:\n", outliers)
```



```
Number of Outliers per Feature:
 Pregnancies        4
Glucose           0
BloodPressure    14
SkinThickness    87
Insulin         159
BMI               8
Pedigree         29
Age               9
Outcome           0
dtype: int64
```

In [9]:

```python
# Initialize KNN
knn = KNeighborsClassifier(n_neighbors=5)  # you can change k
knn.fit(X_train, Y_train)
```

Out[9]:

```
▾ KNeighborsClassifier

KNeighborsClassifier()
```

In [10]:

```python
# Predictions
knn_pred = knn.predict(X_test)
# Metrics
cm = confusion_matrix(Y_test, knn_pred)
accuracy = accuracy_score(Y_test, knn_pred)
error_rate = 1 - accuracy
precision = precision_score(Y_test, knn_pred)
recall = recall_score(Y_test, knn_pred)
f1 = f1_score(Y_test, knn_pred)
# Print results
print("Confusion Matrix:\n", cm)
print("Accuracy Score:", accuracy)
print("Error Rate:", error_rate)
print("Precision Score:", precision)
print("Recall Score:", recall)
print("F1 Score:", f1)
```

```
Confusion Matrix:
 [[88 19]
 [19 28]]
Accuracy Score: 0.7532467532467533
Error Rate: 0.24675324675324672
Precision Score: 0.5957446808510638
Recall Score: 0.5957446808510638
F1 Score: 0.5957446808510638
```

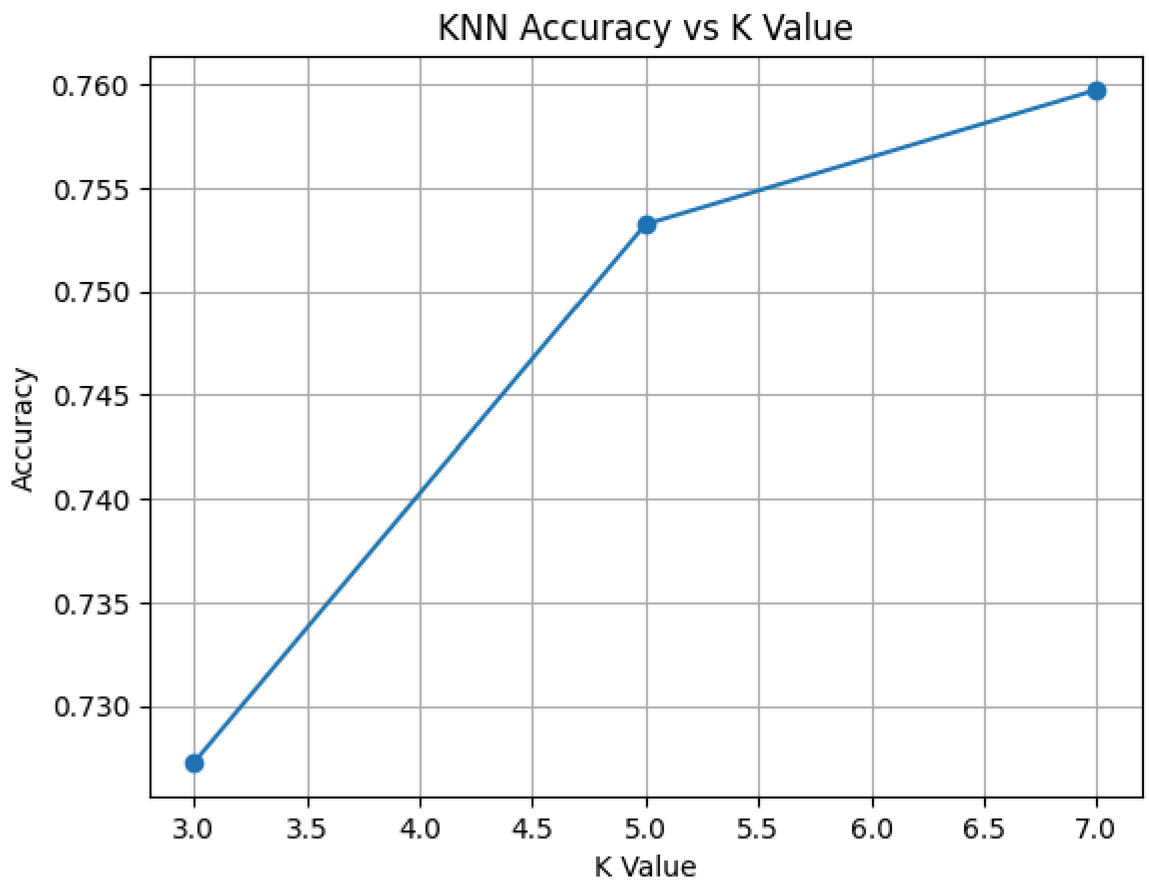
In [11]:

```
accuracy_scores = []

for k in [3, 5, 7]:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, Y_train)
    knn_pred = knn.predict(X_test)
    acc = accuracy_score(Y_test, knn_pred)
    accuracy_scores.append(acc)
    print(f"K = {k} → Accuracy = {acc * 100:.2f}%")

plt.plot([3, 5, 7], accuracy_scores, marker='o')
plt.title("KNN Accuracy vs K Value")
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

```
K = 3 → Accuracy = 72.73%
K = 5 → Accuracy = 75.32%
K = 7 → Accuracy = 75.97%
```



In [ ]: