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

```
df=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/diabetes.csv')
```

```
df
```

	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 × 9 columns

```
df.shape
```

```
(768, 9)
```

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outc
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000

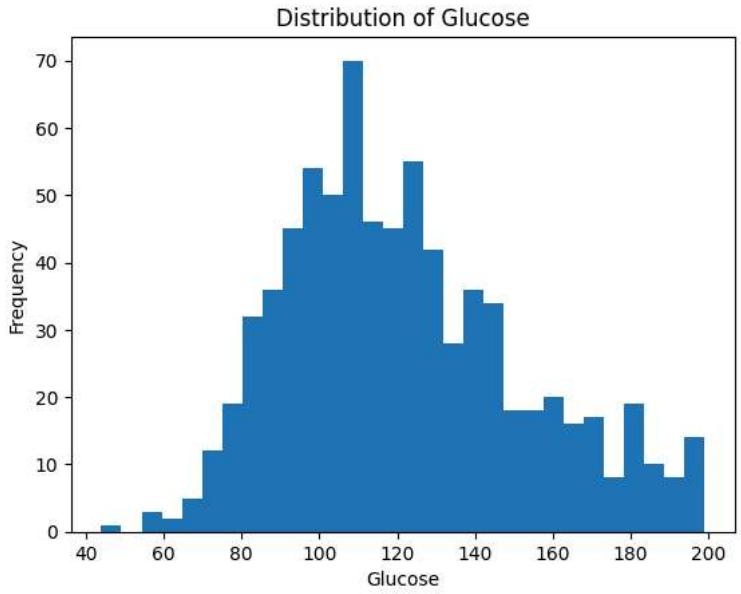
```
missing_cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

for col in missing_cols:
    df[col] = df[col].replace(0, np.nan)  # Replace 0 with np.nan (null) in each specified column
```

```
df.isnull().sum()
```

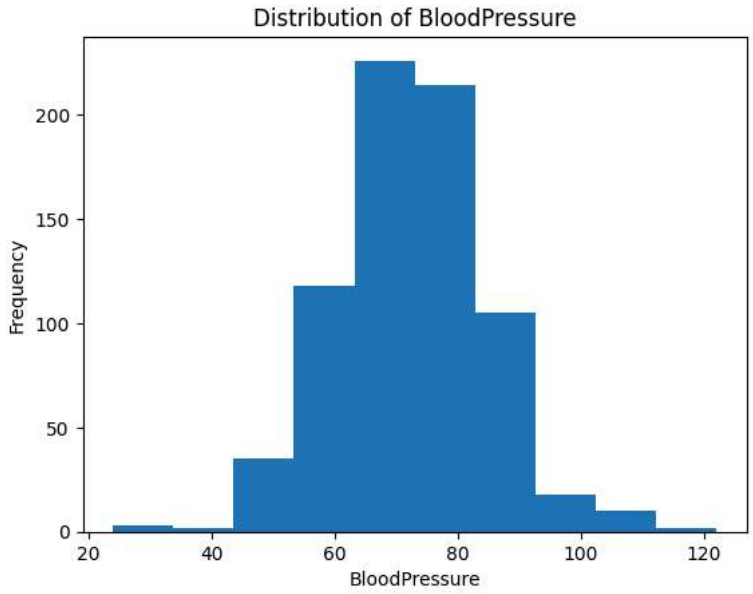
	0
Pregnancies	0
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

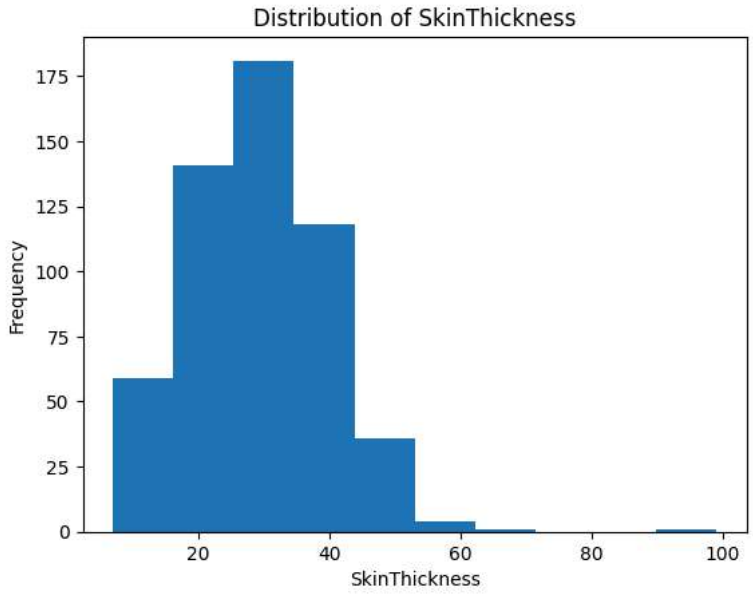
```
plt.hist(df['Glucose'].dropna(), bins=30)
plt.title(f"Distribution of Glucose")
plt.xlabel("Glucose")
plt.ylabel("Frequency")
plt.grid(False)
plt.show()
```



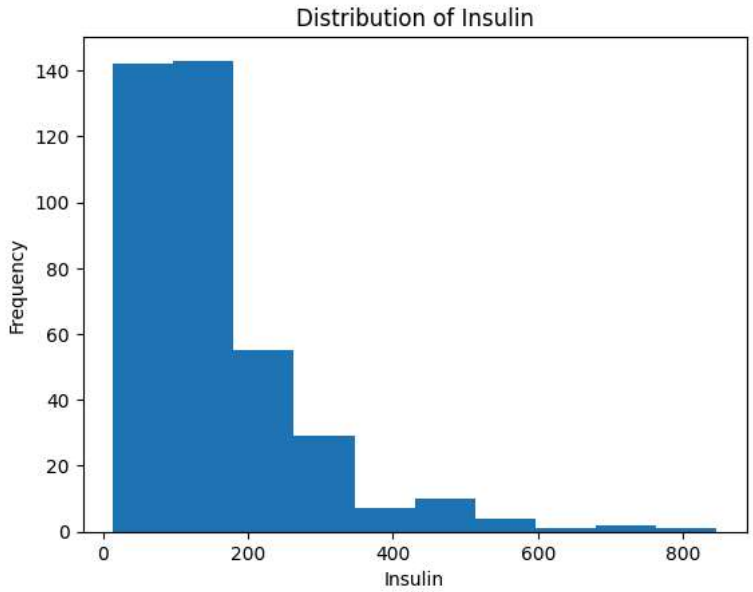

```
plt.hist(df['BloodPressure'].dropna(), bins=10)
plt.title(f"Distribution of BloodPressure")
plt.xlabel("BloodPressure")
plt.ylabel("Frequency")
plt.grid(False)
plt.show()
```



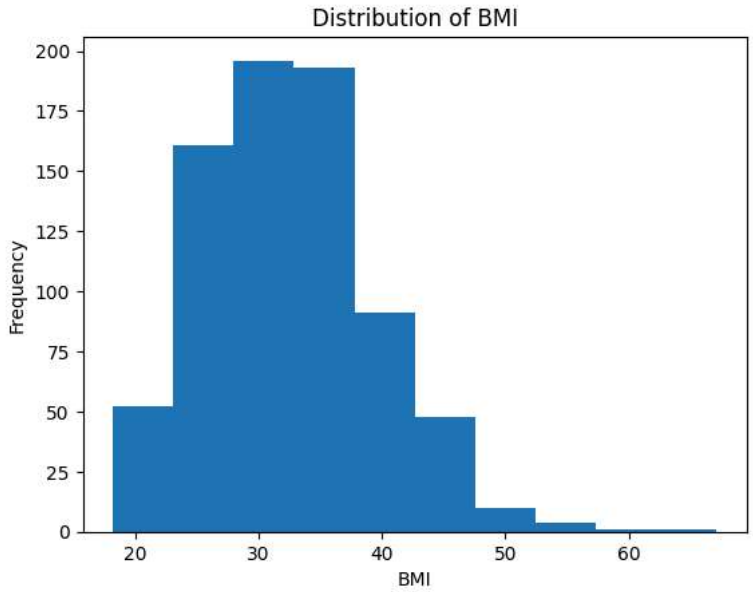

```
plt.hist(df['SkinThickness'].dropna(), bins=10)
plt.title(f"Distribution of SkinThickness")
plt.xlabel("SkinThickness")
plt.ylabel("Frequency")
plt.grid(False)
plt.show()
```




```
plt.hist(df['Insulin'].dropna(), bins=10)
plt.title(f"Distribution of Insulin")
plt.xlabel("Insulin")
plt.ylabel("Frequency")
plt.grid(False)
plt.show()
```




```
plt.hist(df['BMI'].dropna(), bins=10)
plt.title(f"Distribution of BMI")
plt.xlabel("BMI")
plt.ylabel("Frequency")
plt.grid(False)
plt.show()
```




```
df['Glucose'] = df['Glucose'].fillna(df['Glucose'].mean())                    # Replace missing values in the 'Glucose' colum
```

```
df['BloodPressure'] = df['BloodPressure'].fillna(df['BloodPressure'].mean())    # Replace missing values in the 'BloodPressure'
```

```
df['SkinThickness'] = df['SkinThickness'].fillna(df['SkinThickness'].median())    # Replace missing values in the 'SkinThicknes
```

```
df['Insulin'] = df['Insulin'].fillna(df['Insulin'].median())                    # Replace missing values in the 'Insulin' col
```

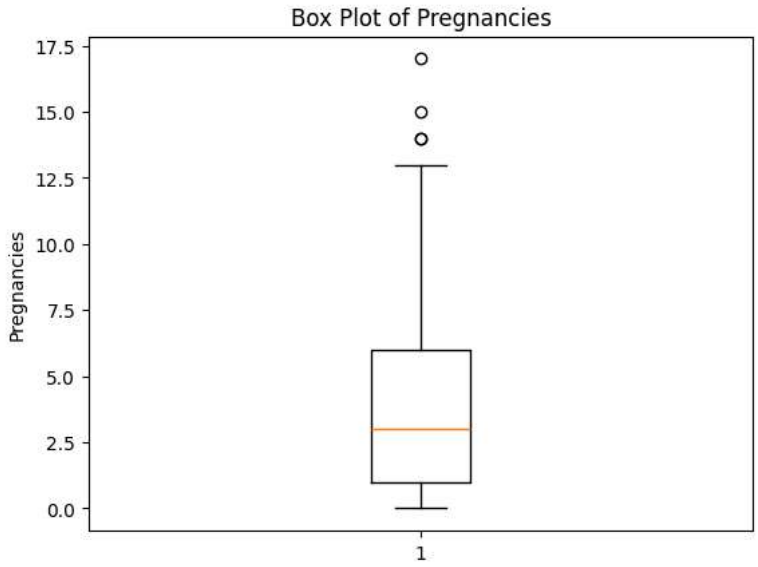
```
df['BMI'] = df['BMI'].fillna(df['BMI'].mean())                                # Replace missing values in the 'BMI' column wi
```

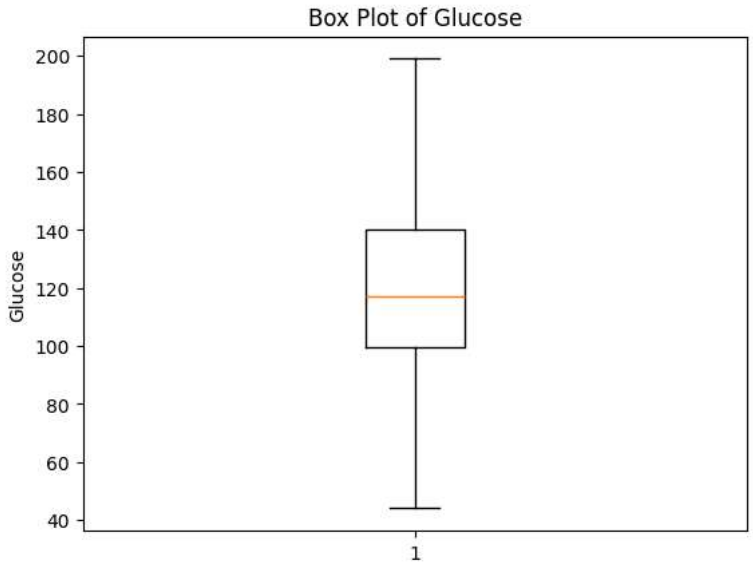
```
df.isnull().sum()
```

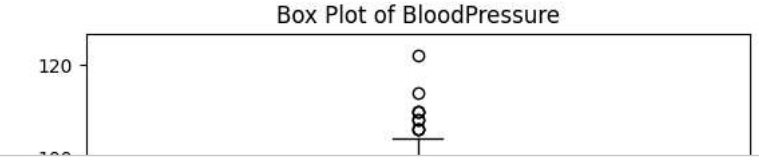
	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

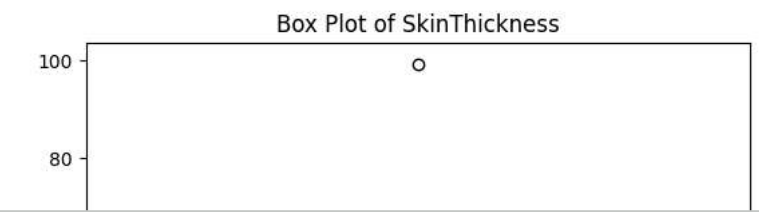
```
numerical_cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Ag

for col in numerical_cols:
    plt.boxplot(df[col].dropna(), vert=True)
    plt.title(f"Box Plot of {col}")
    plt.ylabel(col)
    plt.grid(False)
    plt.show()
```








```
numerical_cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Ag

for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"Outliers in {col}:")
    display(outliers)
```



Outliers in Pregnancies:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
88	15	136.0	70.0	32.0	110.0	37.1	0.153	43	1
159	17	163.0	72.0	41.0	114.0	40.9	0.817	47	1
298	14	100.0	78.0	25.0	184.0	36.6	0.412	46	1
455	14	175.0	62.0	30.0	125.0	33.6	0.212	38	1

Outliers in Glucose:

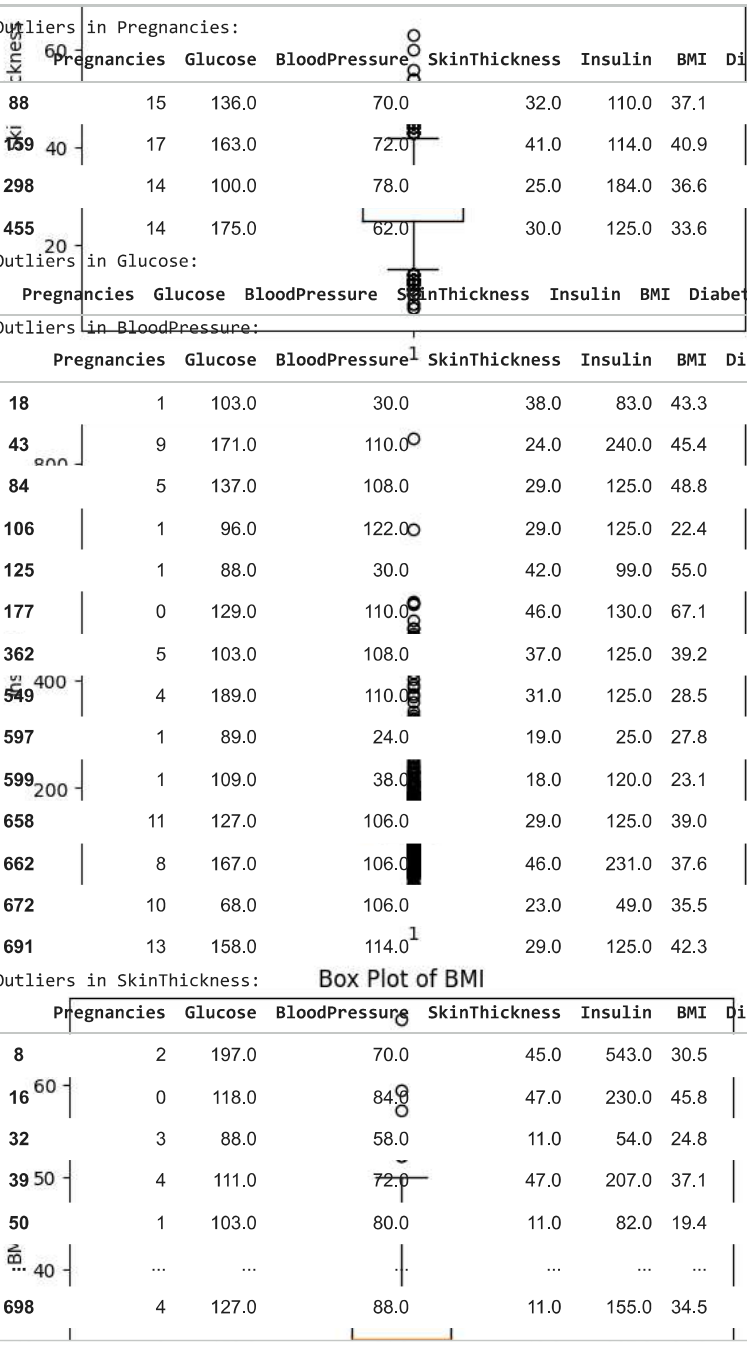
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

Outliers in BloodPressure:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
18	1	103.0	30.0	38.0	83.0	43.3	0.183	33	0
43	9	171.0	110.0	24.0	240.0	45.4	0.721	54	1
84	5	137.0	108.0	29.0	125.0	48.8	0.227	37	1
106	1	96.0	122.0	29.0	125.0	22.4	0.207	27	0
125	1	88.0	30.0	42.0	99.0	55.0	0.496	26	1
177	0	129.0	110.0	46.0	130.0	67.1	0.319	26	1
362	5	103.0	108.0	37.0	125.0	39.2	0.305	65	0
549	4	189.0	110.0	31.0	125.0	28.5	0.680	37	0
597	1	89.0	24.0	19.0	25.0	27.8	0.559	21	0
599	1	109.0	38.0	18.0	120.0	23.1	0.407	26	0
658	11	127.0	106.0	29.0	125.0	39.0	0.190	51	0
662	8	167.0	106.0	46.0	231.0	37.6	0.165	43	1
672	10	68.0	106.0	23.0	49.0	35.5	0.285	47	0
691	13	158.0	114.0	29.0	125.0	42.3	0.257	44	1

Outliers in SkinThickness:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53	1
16	0	118.0	84.0	47.0	230.0	45.8	0.551	31	1
32	3	88.0	58.0	11.0	54.0	24.8	0.267	22	0
39	4	111.0	72.0	47.0	207.0	37.1	1.390	56	1
50	1	103.0	80.0	11.0	82.0	19.4	0.491	22	0
...	...	...	...	...	...	...	...	...	...
698	4	127.0	88.0	11.0	155.0	34.5	0.598	28	0

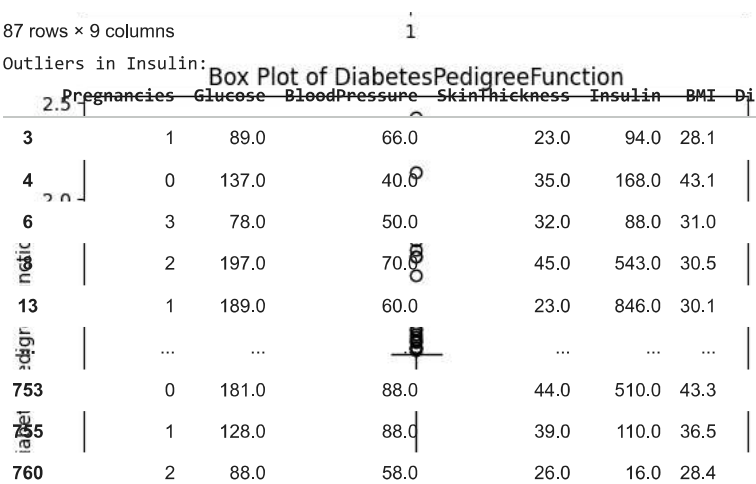


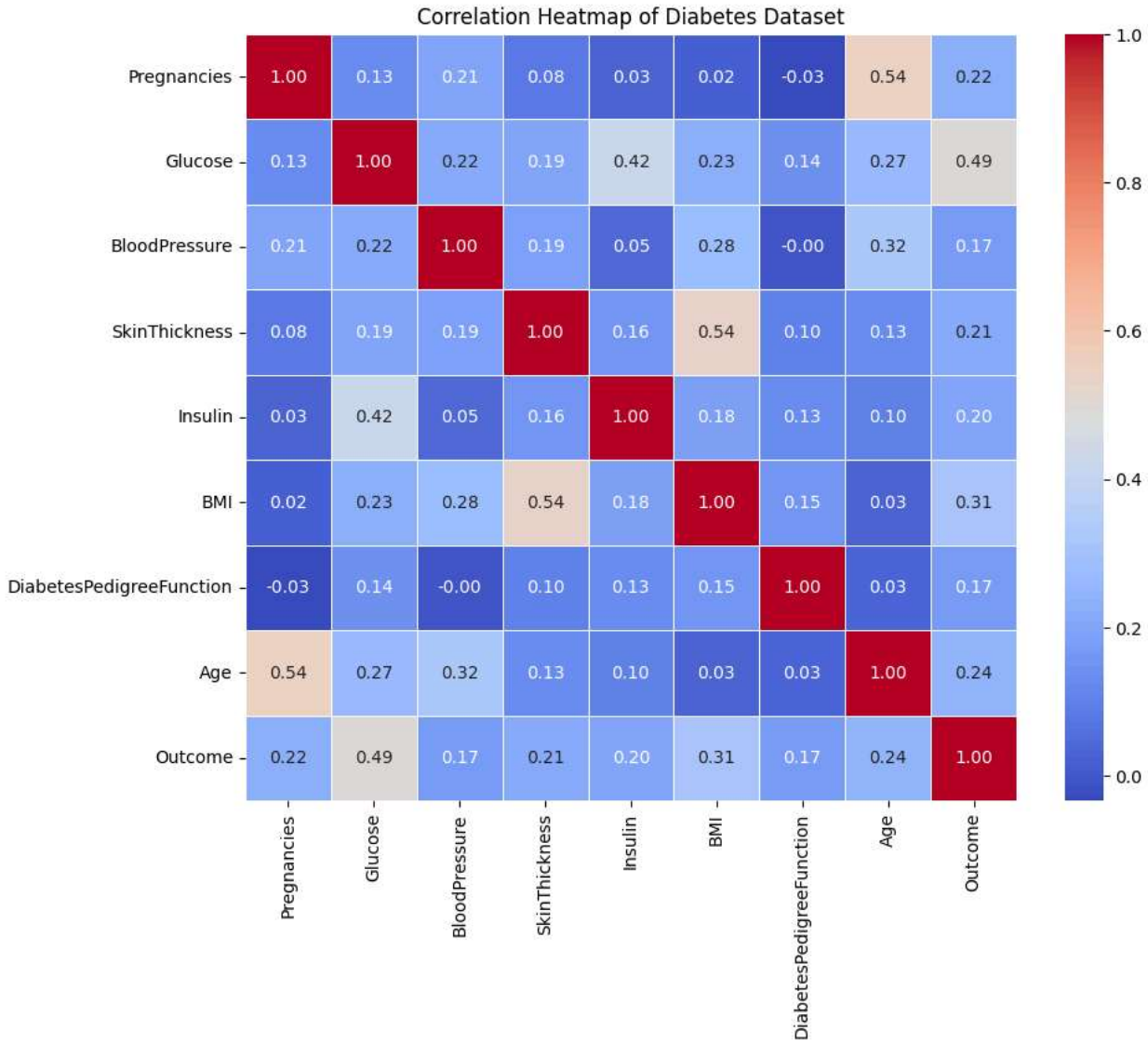

```
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap of Diabetes Dataset')
plt.show()
```

87 rows × 9 columns

Outliers in Insulin:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26	1
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53	1
13	1	189.0	60.0	23.0	846.0	30.1	0.398	59	1
...	...	...	...	...	...	...	...	...	...
753	0	181.0	88.0	44.0	510.0	43.3	0.222	26	1
755	1	128.0	88.0	39.0	110.0	36.5	1.057	37	1
760	2	88.0	58.0	26.0	16.0	28.4	0.766	22	0





292	2	128.0	78.0	37.0	182.0	43.300000		1.224	31	1

```
df.isnull().sum()
```

330	8	118.0	72.0	19.0	125.0	23.100000		1.476	46	0
			82.0	48.0	465.0	38.400000		2.137	25	1
371	0	118.0	64.0	23.0	89.0	32.457464		1.731	21	0
			62.0	18.0	59.0	25.100000		1.268	25	0
395	2	127.0	58.0	24.0	275.0	27.700000		1.600	25	0
			78.0	63.0	14.0	59.400000		2.420	25	1
534	1	77.0	56.0	30.0	56.0	33.300000		1.251	24	0
			52.0	22.0	115.0	28.500000		1.699	25	0
606	1	181.0	78.0	42.0	293.0	40.000000		1.258	22	1
			82.0	24.0	125.0	28.200000		1.282	50	1
621	2	92.0	76.0	20.0	125.0	24.200000		1.698	28	0
								1.461	45	0

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

```
df.to_csv('cleaned_dataset.csv', index=False)                    # Save the cleaned dataframe to a new CSV file
```

661	1	199.0	76.0	43.0	125.0	42.900000		1.394	22	1

```
from sklearn.linear_model import LogisticRegression       # For logistic regression model
from sklearn.tree import DecisionTreeClassifier           # For decision tree model
from sklearn.ensemble import RandomForestClassifier       # For random forest model
from sklearn.svm import SVC                               # For support vector machine
```

```
import matplotlib.pyplot as plt                          # Import pyplot for plotting
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_auc_score        #
```

```
# Split dataframe into features (X) and label (y)
X = df.drop('Outcome', axis=1)                                          # Select all columns except 'Outcome' as predic
y = df['Outcome']                                                       # Select the 'Outcome' column as the target var
```

```
666  4  145.0  82.000000  18.0  125.0  32.500000  0.235  70  1
```

```
from sklearn.model_selection import train_test_split                    # Import the function to split your dataset int
# X = features, y = label/outcome, Split the features (X) and labels (y) into training and test data; test_size=0.2 reserves 20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
# Logistic Regression
logreg = LogisticRegression(max_iter=200)                               # Create a Logistic Regression classifier with
logreg.fit(X_train, y_train)                                            # Train the logistic regression model using the
```

```
LogisticRegression
LogisticRegression(max_iter=200)
```

```
# Decision Tree
dtree = DecisionTreeClassifier(random_state=42)                         # Initialize a decision tree classifier with a
dtree.fit(X_train, y_train)                                             # Train the decision tree model using the train
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)          # Initialize a random forest with 100 trees and
rf.fit(X_train, y_train)                                                # Train the random forest model using the train
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Support Vector Machine (SVM)
svm = SVC(probability=True, random_state=42)                            # Initialize a support vector machine classifie
svm.fit(X_train, y_train)                                               # Train the SVM model using the training data
```

```
SVC
SVC(probability=True, random_state=42)
```

```
def eval_metrics(y_test, y_pred, model_name):                           # This function prints accuracy, precision, rec
    print(f"\n{model_name} Results:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
from sklearn.neighbors import KNeighborsClassifier                      # Import the k-nearest neighbors classifier for

# Initialize and fit k-NN model (using k=5 neighbors)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predict on test set and evaluate
y_pred_knn = knn.predict(X_test)
eval_metrics(y_test, y_pred_knn, "k-Nearest Neighbors")
```

```
k-Nearest Neighbors Results:
Accuracy: 0.6818181818181818
Precision: 0.5510204081632653
Recall: 0.5
F1 Score: 0.5242718446601942
Confusion Matrix:
 [[78 22]
 [27 27]]
```

```
from sklearn.naive_bayes import GaussianNB                                    # Import the Gaussian Naive Bayes classifier fo

# Initialize and fit Naive Bayes model
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Predict on test set and evaluate
y_pred_gnb = gnb.predict(X_test)
eval_metrics(y_test, y_pred_gnb, "Naive Bayes (GaussianNB)")
```

```
Naive Bayes (GaussianNB) Results:
Accuracy: 0.7012987012987013
Precision: 0.5666666666666667
Recall: 0.6296296296296297
F1 Score: 0.5964912280701754
Confusion Matrix:
 [[74 26]
 [20 34]]
```

```
from xgboost import XGBClassifier                                             # Import XGBoost, a powerful and efficient grad

# Initialize and fit XGBoost model
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)

# Predict on test set and evaluate
y_pred_xgb = xgb.predict(X_test)
eval_metrics(y_test, y_pred_xgb, "XGBoost")
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [07:47:52] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

XGBoost Results:
Accuracy: 0.7597402597402597
Precision: 0.660377358490566
Recall: 0.6481481481481481
F1 Score: 0.6542056074766355
Confusion Matrix:
 [[82 18]
 [19 35]]
```

```
y_pred_logreg = logreg.predict(X_test)                                        # Variable that stores Logistic Regression pred
```

```
y_pred_dtree = dtree.predict(X_test)                                          # Variable that stores Decision Tree prediction
```

```
y_pred_rf = rf.predict(X_test)                                                # Variable that stores Random Forest prediction
```

```
y_pred_svm = svm.predict(X_test)                                              # Variable that stores SVM predictions
```

```
eval_metrics(y_test, y_pred_logreg, "Logistic Regression")                    # Display and print evaluation results for Logi
```

```
Logistic Regression Results:
Accuracy: 0.7012987012987013
Precision: 0.5869565217391305
Recall: 0.5
F1 Score: 0.54
Confusion Matrix:
 [[81 19]
 [27 27]]
```

```
eval_metrics(y_test, y_pred_dtree, "Decision Tree")                           # Display and print evaluation results for Deci
```

```
Decision Tree Results:
Accuracy: 0.6688311688311688
Precision: 0.5333333333333333
Recall: 0.4444444444444444
F1 Score: 0.48484848484848486
Confusion Matrix:
 [[79 21]
```

```
 [30 24]]
```

```
eval_metrics(y_test, y_pred_rf, "Random Forest")                    # Display and print evaluation results for Rand
```

```
Random Forest Results:
Accuracy: 0.7597402597402597
Precision: 0.6808510638297872
Recall: 0.5925925925925926
F1 Score: 0.6336633663366337
Confusion Matrix:
 [[85 15]
 [22 32]]
```

```
eval_metrics(y_test, y_pred_svm, "SVM")                             # Display and print evaluation results for SVM
```

```
SVM Results:
Accuracy: 0.7207792207792207
Precision: 0.6486486486486487
Recall: 0.4444444444444444
F1 Score: 0.5274725274725275
Confusion Matrix:
 [[87 13]
 [30 24]]
```

```
from sklearn.model_selection import cross_val_score

# Cross-Validation for Logistic Regression
logreg_cv = cross_val_score(logreg, X, y, cv=5, scoring='accuracy')
print("Logistic Regression Cross-Val Accuracy: {:.3f} (+/- {:.3f})".format(logreg_cv.mean(), logreg_cv.std()))
```

```
Logistic Regression Cross-Val Accuracy: 0.772 (+/- 0.020)
```

```
# Cross-Validation for Decision Tree
dtree_cv = cross_val_score(dtree, X, y, cv=5, scoring='accuracy')
print("Decision Tree Cross-Val Accuracy: {:.3f} (+/- {:.3f})".format(dtree_cv.mean(), dtree_cv.std()))
```

```
Decision Tree Cross-Val Accuracy: 0.720 (+/- 0.041)
```

```
# Cross-Validation for Random Forest
rf_cv = cross_val_score(rf, X, y, cv=5, scoring='accuracy')
print("Random Forest Cross-Val Accuracy: {:.3f} (+/- {:.3f})".format(rf_cv.mean(), rf_cv.std()))
```

```
Random Forest Cross-Val Accuracy: 0.760 (+/- 0.030)
```

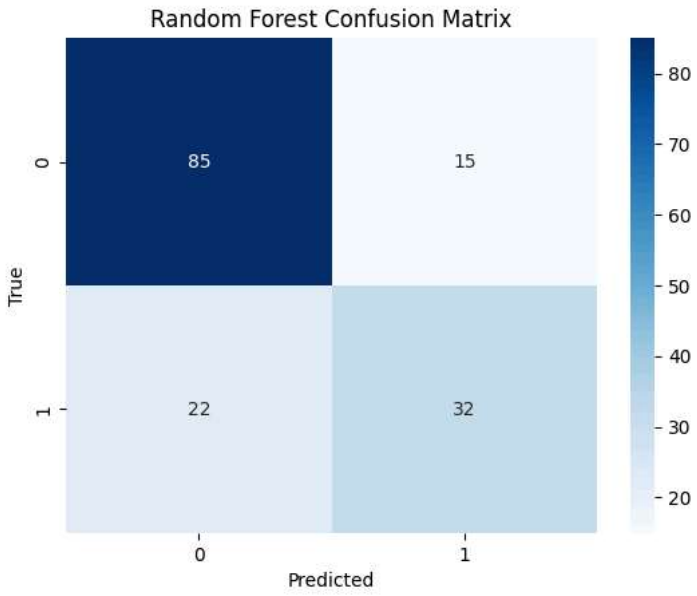
```
# Cross-Validation for SVM
svm_cv = cross_val_score(svm, X, y, cv=5, scoring='accuracy')
print("SVM Cross-Val Accuracy: {:.3f} (+/- {:.3f})".format(svm_cv.mean(), svm_cv.std()))
```

```
SVM Cross-Val Accuracy: 0.760 (+/- 0.023)
```

```
# Random Forest - Performance Metrics
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_precision = precision_score(y_test, y_pred_rf)
rf_recall = recall_score(y_test, y_pred_rf)
rf_f1 = f1_score(y_test, y_pred_rf)
rf_auc = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
print("Random Forest:")
print("Accuracy:", rf_accuracy)
print("Precision:", rf_precision)
print("Recall:", rf_recall)
print("F1 Score:", rf_f1)
print("AUC:", rf_auc)
```

```
Random Forest:
Accuracy: 0.7597402597402597
Precision: 0.6808510638297872
Recall: 0.5925925925925926
F1 Score: 0.6336633663366337
AUC: 0.8200000000000001
```

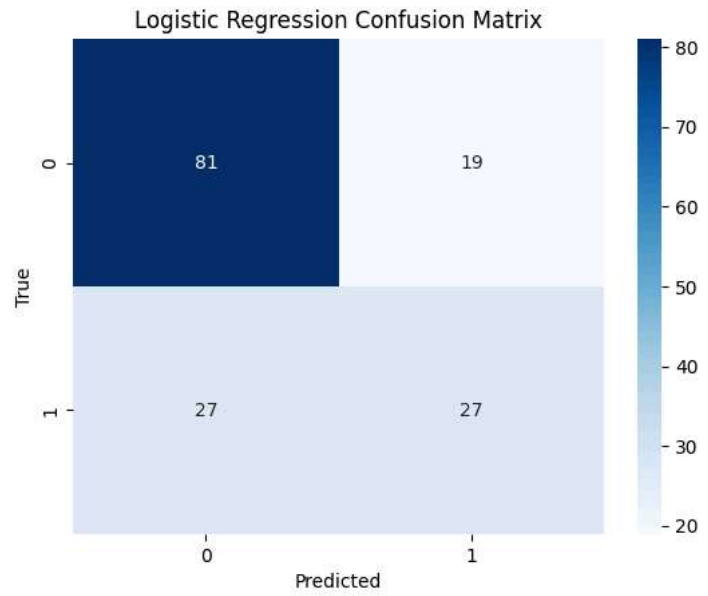
```
import seaborn as sns

# Confusion matrix for Random Forest
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Blues')
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```
# Logistic Regression - Performance Metrics
logreg_accuracy = accuracy_score(y_test, y_pred_logreg)
logreg_precision = precision_score(y_test, y_pred_logreg)
logreg_recall = recall_score(y_test, y_pred_logreg)
logreg_f1 = f1_score(y_test, y_pred_logreg)
logreg_auc = roc_auc_score(y_test, logreg.predict_proba(X_test)[:, 1])
print("Logistic Regression:")
print("Accuracy:", logreg_accuracy)
print("Precision:", logreg_precision)
print("Recall:", logreg_recall)
print("F1 Score:", logreg_f1)
print("AUC:", logreg_auc)
```

```
Logistic Regression:
Accuracy: 0.7012987012987013
Precision: 0.5869565217391305
Recall: 0.5
F1 Score: 0.54
AUC: 0.8127777777777777
```

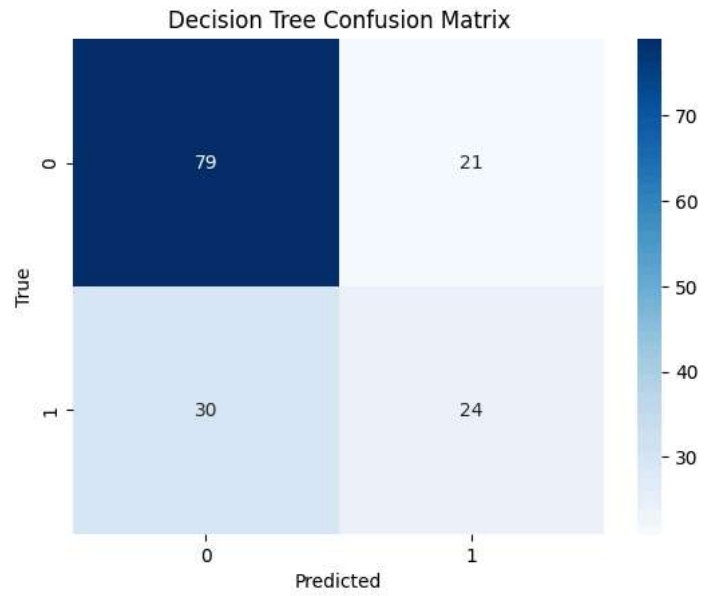
```
# Confusion matrix for Logistic Regression
sns.heatmap(confusion_matrix(y_test, y_pred_logreg), annot=True, fmt='d', cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```
# Decision Tree - Performance Metrics
dtree_accuracy = accuracy_score(y_test, y_pred_dtree)
dtree_precision = precision_score(y_test, y_pred_dtree)
dtree_recall = recall_score(y_test, y_pred_dtree)
dtree_f1 = f1_score(y_test, y_pred_dtree)
dtree_auc = roc_auc_score(y_test, dtree.predict_proba(X_test)[:, 1])
print("Decision Tree:")
print("Accuracy:", dtree_accuracy)
print("Precision:", dtree_precision)
print("Recall:", dtree_recall)
print("F1 Score:", dtree_f1)
print("AUC:", dtree_auc)
```

```
Decision Tree:
Accuracy: 0.6688311688311688
Precision: 0.5333333333333333
Recall: 0.4444444444444444
F1 Score: 0.48484848484848486
AUC: 0.6172222222222222
```

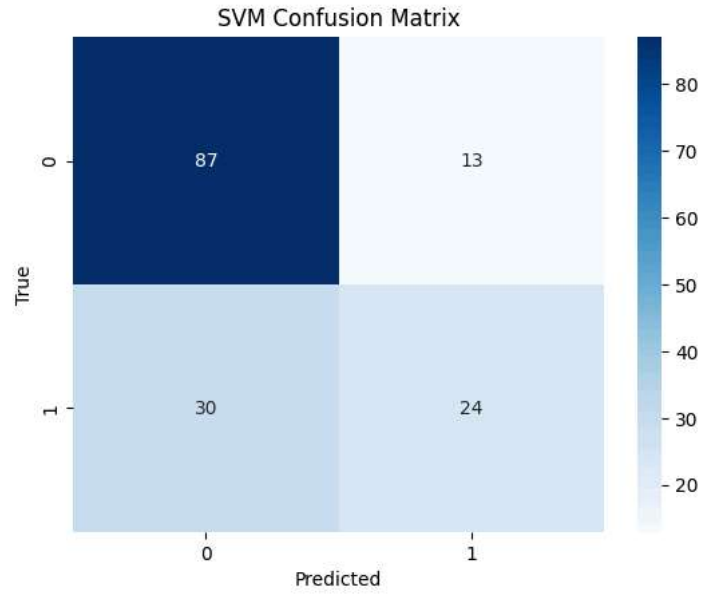
```
# Confusion matrix for Decision Tree
sns.heatmap(confusion_matrix(y_test, y_pred_dtree), annot=True, fmt='d', cmap='Blues')
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



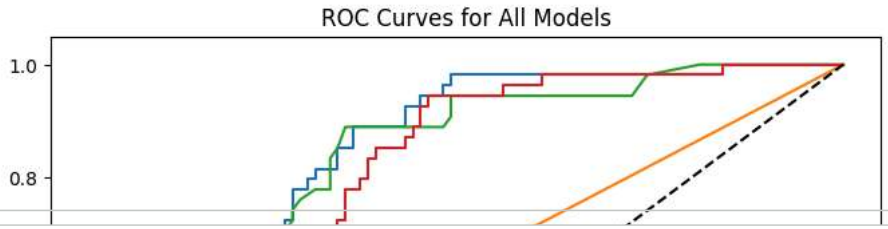

```
# SVM - Performance Metrics
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_precision = precision_score(y_test, y_pred_svm)
svm_recall = recall_score(y_test, y_pred_svm)
svm_f1 = f1_score(y_test, y_pred_svm)
svm_auc = roc_auc_score(y_test, svm.predict_proba(X_test)[:, 1])
print("Support Vector Machine (SVM):")
print("Accuracy:", svm_accuracy)
print("Precision:", svm_precision)
print("Recall:", svm_recall)
print("F1 Score:", svm_f1)
print("AUC:", svm_auc)
```

```
Support Vector Machine (SVM):
Accuracy: 0.7207792207792207
Precision: 0.6486486486486487
Recall: 0.4444444444444444
F1 Score: 0.5274725274725275
AUC: 0.7811111111111111
```

```
# Confusion matrix for Support Vector Machine (SVM)
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d', cmap='Blues')
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```
# Plot the ROC curves and AUC values for all models to visually compare their ability to distinguish between classes.
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

plt.figure(figsize=(8,6))
models = [
    (logreg, y_pred_logreg, "Logistic Regression"),
    (dtree, y_pred_dtree, "Decision Tree"),
    (rf, y_pred_rf, "Random Forest"),
    (svm, y_pred_svm, "SVM")
]

# Plot ROC curve for each model
for model, y_pred, name in models:
    # Get predicted probabilities for the positive class (1)
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)[:,1]
    else:
        y_score = model.decision_function(X_test)  # For SVM without probability
    fpr, tpr, _ = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal dashed line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for All Models')
plt.legend(loc='lower right')
plt.show()
```




```
import pandas as pd

def predict_diabetes_from_input():
    print("Enter the following details:")

    pregnancies = int(input("Pregnancies: "))
    glucose = float(input("Glucose: "))
    bloodpressure = float(input("BloodPressure: "))
    skinthickness = float(input("SkinThickness: "))
    insulin = float(input("Insulin: "))
    bmi = float(input("BMI: "))
    dpf = float(input("DiabetesPedigreeFunction: "))
    age = int(input("Age: "))
```