Diabetes Classification

Louis Bailey

Introduction:

About the Data:This dataset can be found here on the UCI Machine Learning Repository website. It contains signs and symptpoms data of newly diabetic or would be diabetic patients. It was collected using direct questionnaires from the patients of Sylhet Diabetes Hospital in Sylhet, Bangladesh.

Project Goal: The goal of the project is to create a classification model that can predict positive/negative outcomes of diabetes. More specifically, the goal will be to build a model that can correctly identify the highest amount of positive cases while maintaining a reasonably low false positive rate.

Contents

Libraries

In [1]:
```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns

from imblearn.pipeline import make_pipeline

from sklearn.model_selection import cross_val_score, GridSearchCV, StratifiedKFold, train_test_split
from sklearn.metrics import classification_report, accuracy_score, roc_curve, roc_auc_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.inspection import permutation_importance

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
```

Import Data

In [2]:
```
df = pd.read_csv('uci_diabetes_data.csv')
df.shape
```

Out[2]:
```
(520, 17)
```

In [3]:
```
df.head()
```

Out[3]:

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity	cl
0	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Posit
1	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	No	Yes	No	Posit
2	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Posit
3	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Posit
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Posit

1 | Clean

Changing Column Names

In [4]:
```
mappings = {'sudden weight loss':'sudden_weight_loss', 'Genital thrush':'Genital_thrush',
            'visual blurring':'visual_blurring', 'delayed healing':'delayed_healing',
            'partial paresis':'partial_paresis','muscle stiffness':'muscle_stiffness',
            'class':'class_'}

df.rename(columns=mappings, inplace=True)
```

Unique Values For Each Column

In [5]:
```
for col in df.columns:
    print(df[col].unique())
```

```
[40 58 41 45 60 55 57 66 67 70 44 38 35 61 54 43 62 39 48 32 42 52 53 37
 49 63 30 50 46 36 51 59 65 25 47 28 68 56 31 85 90 72 69 79 34 16 33 64
 27 29 26]
['Male' 'Female']
['No' 'Yes']
['Yes' 'No']
['No' 'Yes']
['Yes' 'No']
['No' 'Yes']
['No' 'Yes']
['No' 'Yes']
['Yes' 'No']
['No' 'Yes']
['Yes' 'No']
['No' 'Yes']
['Yes' 'No']
['Yes' 'No']
['Yes' 'No']
['Positive' 'Negative']
```

> There doesn't appear to be any errors in the the columns' values.

Missing & Duplicate Values

In [6]:
```
print(f'Missing:{df.isna().sum().sum()}')
print(f'Duplicates:{df.duplicated().sum()}')
```

```
Missing:0
Duplicates:269
```

> There are no missing values.
>
> There are 269 duplicate values. There doesn't appear to be any information on the UCI Repository website about these duplicates or about any resampling techniques being done on the data, so these values will be dropped.

In [7]:
```
df = df.drop_duplicates()
print(f'Duplicates:{df.duplicated().sum()}')
print(f'Shape:{df.shape}')
```

```
Duplicates:0
Shape:(251, 17)
```

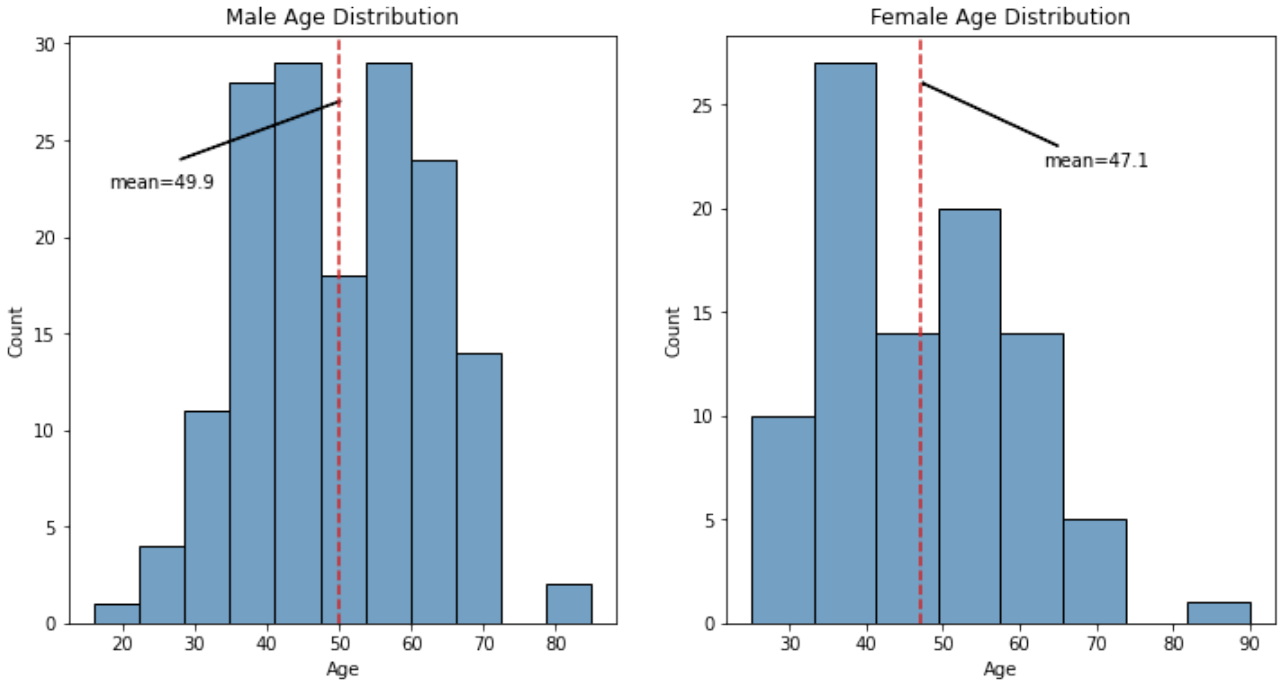
> Now there are no duplicates and the data has a shape of 251x17

---

2 | Data Exploration

Age Distributions by Gender

In [8]:
```
#
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

sns.histplot(df[df.Gender == 'Male'].Age, ax=ax1, color='steelblue', legend=False)
ax1.set_title('Male Age Distribution')
ax1.axvline(x=np.mean(df[df.Gender == 'Male'].Age), color='tab:red', linestyle='--')
ax1.arrow(28, 24, 22, 3, width = 0.05, color='black')
ax1.text(18,22.5, 'mean=49.9')

sns.histplot(df[df.Gender == 'Female'].Age, ax=ax2, color='steelblue')
ax2.set_title('Female Age Distribution')
ax2.axvline(x=np.mean(df[df.Gender == 'Female'].Age), color='tab:red', linestyle='--')
ax2.arrow(65, 23, -17.5, 3, width = 0.05, color='black')
ax2.text(63,22, 'mean=47.1')

plt.show()
```

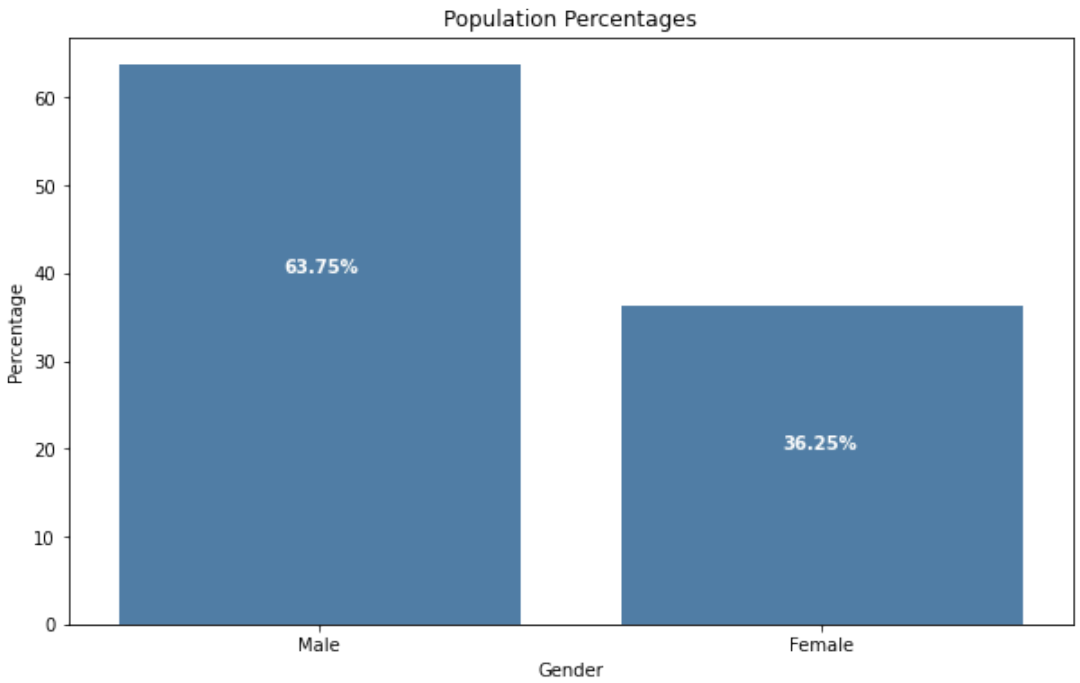


> The mean age for males is slightly higher at 49.9 years old.

Population Percentages by Gender

In [9]:
```
#
plt.figure(figsize=(10,6))

class_percentages = df.Gender.value_counts()/len(df.Gender) * 100
sns.barplot(x=class_percentages.index, y=class_percentages.values, color='steelblue')

p1 = round(class_percentages.values[0],2)
p2 = round(class_percentages.values[1],2)

plt.text(-0.07,40,f'{p1}%',color='w', fontweight='bold')
plt.text(0.92,20,f'{p2}%',color='w', fontweight='bold')

plt.title('Population Percentages')
plt.xlabel('Gender')
plt.ylabel('Percentage')
```

```
plt.show()
```

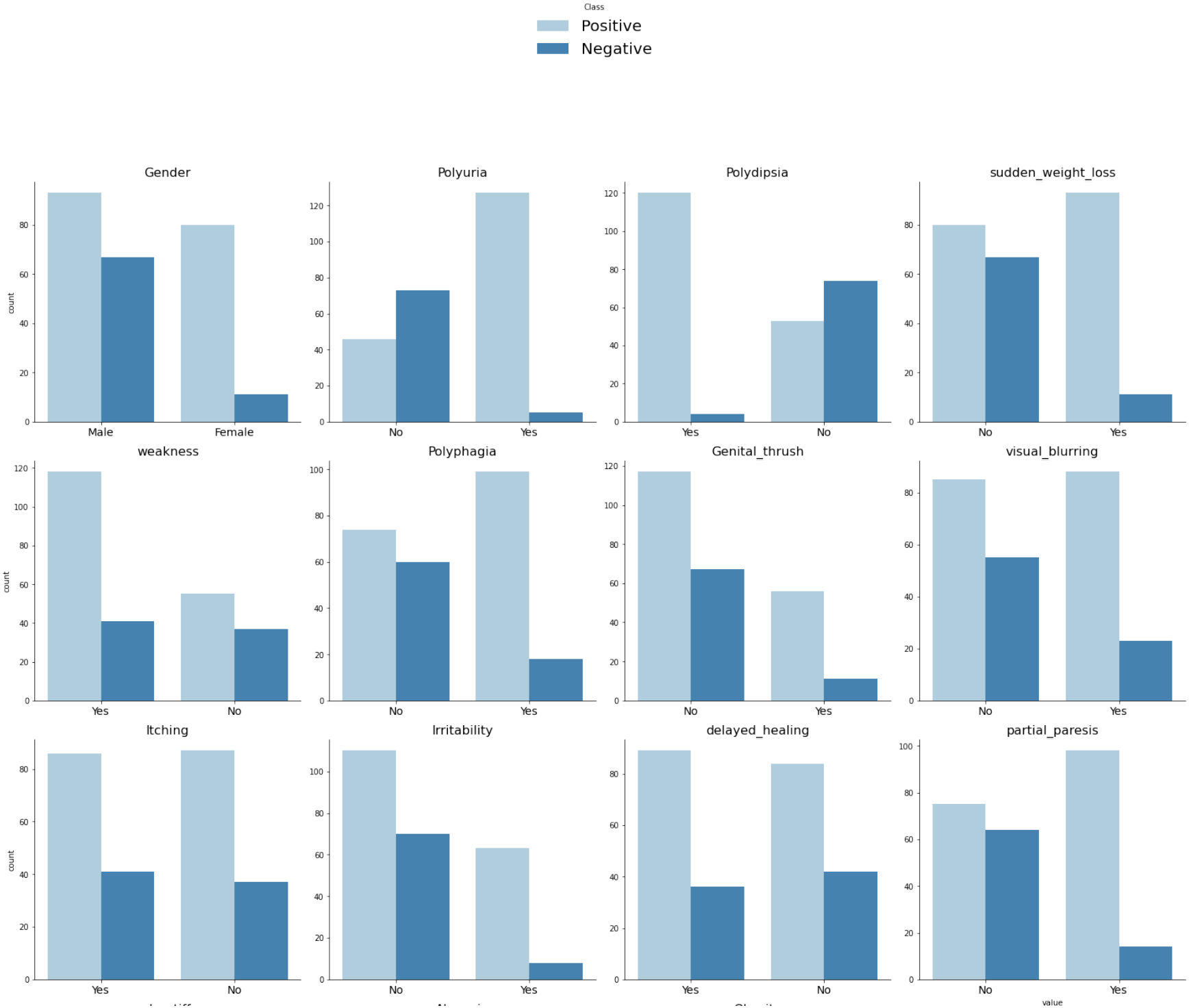


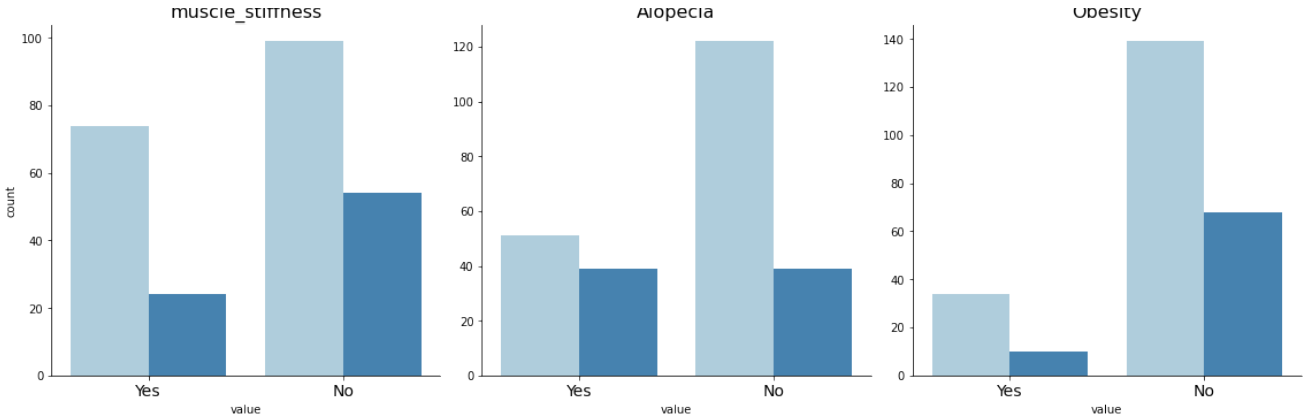
> The majority of the population, about 64%, is male.

Box Plots of Negative and Positive Outcomes by Category

In [10]:
```
#
my_vars = df.drop(['class_', 'Age'], axis=1).columns

df_melted = df.melt(id_vars='class_', value_vars=my_vars, var_name='variable', value_name='value')

g = sns.catplot(data=df_melted, x='value', hue='class_', col='variable', kind='count', col_wrap=4, sharey=False, sharex=False,
                palette="Blues")
g.set_titles('{col_name}', size=16)
g.set_xticklabels(size=14)

sns.move_legend(g, title='Class', loc='upper center', bbox_to_anchor=(0.5, 1.15), fontsize=20)
```

```
plt.tight_layout(pad=1.0)
plt.show()
```

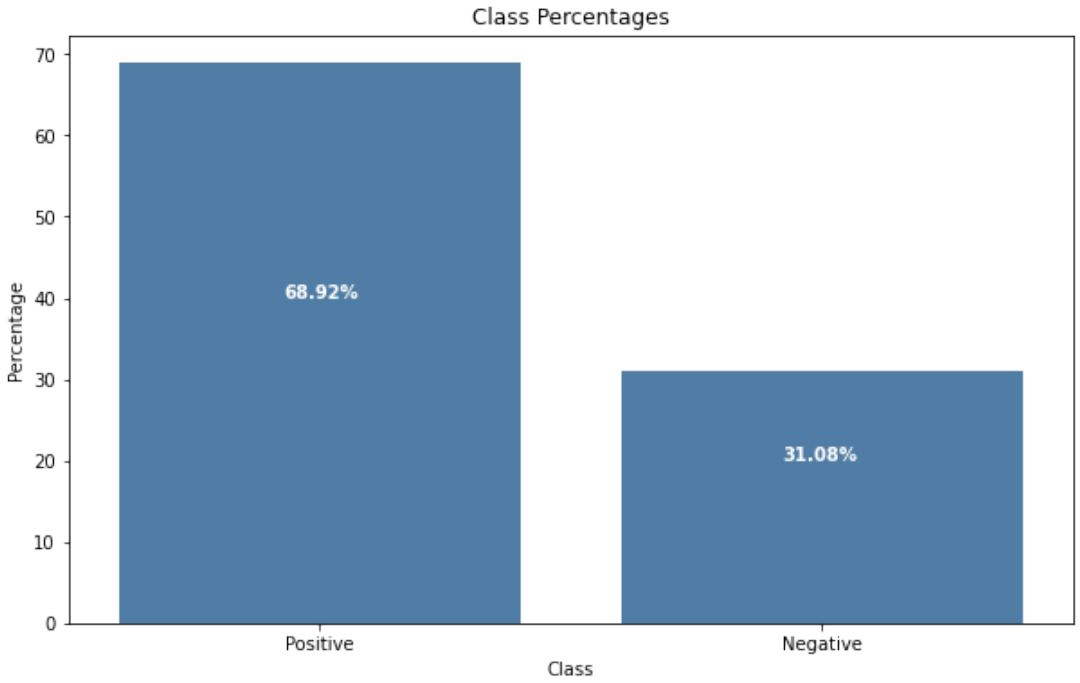

Class
Positive
Negative
Gender
Male
Female
Polyuria
No
Yes
Polydipsia
Yes
No
sudden_weight_loss
No
Yes
weakness
Yes
No
Polyphagia
No
Yes
Genital_thrush
No
Yes
visual_blurring
No
Yes
Itching
Yes
No
Irritability
No
Yes
delayed_healing
Yes
No
partial_paresis
No
Yes
count
value



> From these plots it is clear what predictor variables are likely to be strongest. For Gender_Female, Polyuria_yes, Polydispia_yes, sudden_weight_loss_yes, weakness_yes and partial_paresis_yes the vast majority testsed positive.
>
> These numbers being so different make it seem like a well fitting model will not be too dificult to create here.

Percentage of Negative & Positive Outcomes

In [11]:
```
#
plt.figure(figsize=(10,6))

class_percentages = df.class_.value_counts()/len(df.class_) * 100
sns.barplot(x=class_percentages.index, y=class_percentages.values, color='steelblue')

p1 = round(class_percentages.values[0],2)
p2 = round(class_percentages.values[1],2)

plt.text(-0.07,40,f'{p1}%',color='w', fontweight='bold')
plt.text(0.92,20,f'{p2}%',color='w', fontweight='bold')

plt.title('Class Percentages')
plt.xlabel('Class')
plt.ylabel('Percentage')

plt.show()
```



Most of the population, about 69%, tested positive for diabetes. As a baseline, if one guessed positive everytime with this data, they would be right about 69% of the time.

---

3 | Variable Transformations

Dummy Variables

In [12]:
```
categorical = ['Gender', 'Polyuria', 'Polydipsia','sudden_weight_loss', 'weakness', 'Polyphagia',
               'Genital_thrush', 'visual_blurring', 'Itching', 'Irritability', 'delayed_healing',
               'partial_paresis', 'muscle_stiffness', 'Alopecia', 'Obesity']
```

In [13]:
```
df = pd.get_dummies(df, columns=categorical, drop_first=True)
```

In [14]:
```
df.head(5)
```

Out[14]:

	Age	class_	Gender_Male	Polyuria_Yes	Polydipsia_Yes	sudden_weight_loss_Yes	weakness_Yes	Polyphagia_Yes	Genital_thrush_Yes	visual_blurring_Yes	Itchin
0	40	Positive	True	False	True	False	True	False	False	False	
1	58	Positive	True	False	False	False	True	False	False	True	
2	41	Positive	True	True	False	False	True	True	False	False	
3	45	Positive	True	False	False	True	True	True	True	False	
4	60	Positive	True	True	True	True	True	True	False	True	

Target Transformation

In [15]:
```
df['class_'] = df['class_'].apply(lambda x: 1 if x=='Positive' else 0)
```

4 | Features, Target and Train/Test Split

In [16]:
```
X = df.drop('class_', axis=1)
y = df.class_
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

5 | Out-of-the-Box Logistic Regression Model

In [18]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred_train = model.predict(X_train)
```

In [19]:
```
print(f'Train Accuracy: {round(accuracy_score(y_train, y_pred_train),2)}')
print(f'Test Accuracy: {round(accuracy_score(y_test, y_pred),2)}')
```

```
Train Accuracy: 0.92
Test Accuracy: 0.84
```

> The untuned logistic regression model already performs well. The accuracy is high, and the train vs test scores do not appear different enough to indicate overfitting.

---

6 | Pipelines & Search Spaces

Pipes

In [20]:
```
logreg_pipeline = make_pipeline(MinMaxScaler(),
                                LogisticRegression(random_state=1))

rf_pipeline     = make_pipeline(RandomForestClassifier(random_state=1))

svc_pipeline    = make_pipeline(MinMaxScaler(),
                                SVC(probability=True, random_state=1))

xgb_pipeline    = make_pipeline(xgb.XGBClassifier(random_state=1))

knn_pipeline    = make_pipeline(MinMaxScaler(),
                                KNeighborsClassifier())
```

Search Spaces

In [21]:
```
search_spaces = {
    'kneighborsclassifier': {'kneighborsclassifier__n_neighbors': [1,2,3,4,5,6,7,8,9,10]},

    'logisticregression': [
        {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100],
         'logisticregression__penalty': ['l1', 'l2'],
         'logisticregression__solver': ['liblinear'],
         'logisticregression__class_weight': [None, 'balanced']},
        {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100],
         'logisticregression__penalty': ['l2'],
         'logisticregression__solver': ['newton-cg', 'lbfgs', 'saga'],
         'logisticregression__class_weight': [None, 'balanced']},
        {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100],
         'logisticregression__penalty': ['elasticnet'],
         'logisticregression__solver': ['saga'],
         'logisticregression__l1_ratio': [0, 0.5, 1],
```

```
        'logisticregression__class_weight': [None, 'balanced']}
    ],

    'randomforestclassifier': {
        'randomforestclassifier__n_estimators': [10, 50, 100, 200],
        'randomforestclassifier__max_depth':[3, 4, 5, 6],
        'randomforestclassifier__min_samples_split':[2, 4, 6, 8],
        'randomforestclassifier__class_weight': [None, 'balanced']},

    'xgbclassifier': {
         'xgbclassifier__n_estimators': [100, 200],
         'xgbclassifier__learning_rate': [0.01, 0.1, 1],
         'xgbclassifier__max_depth': [1,2,3,4],
         'xgbclassifier__min_child_weight': [1, 4]},

    'svc': {
        'svc__C': [0.1, 1, 10, 100, 1000],
        'svc__kernel': ['rbf'],
        'svc__gamma': [0.001, 0.01, 0.1, 1, 'scale', 'auto'],
        'svc__class_weight': [None, 'balanced']}
}
```

Grid Search for Best Hyperparameters

In [22]:

```
knn_search = GridSearchCV(knn_pipeline, search_spaces['kneighborsclassifier'], cv=5,
                          scoring='accuracy', n_jobs=-1).fit(X_train, y_train)

logreg_search = GridSearchCV(logreg_pipeline, search_spaces['logisticregression'], cv=5,
                             scoring='accuracy', n_jobs=-1).fit(X_train, y_train)

rf_search = GridSearchCV(rf_pipeline, search_spaces['randomforestclassifier'], cv=5,
                         scoring='accuracy', n_jobs=-1).fit(X_train, y_train)

xgb_search = GridSearchCV(xgb_pipeline, search_spaces['xgbclassifier'], cv=5,
                          scoring='accuracy', n_jobs=-1).fit(X_train, y_train)

svc_search = GridSearchCV(svc_pipeline, search_spaces['svc'], cv=5,
                          scoring='accuracy', n_jobs=-1).fit(X_train, y_train)

print(f'kNN tuned params: {knn_search.best_params_} \n')
print(f'Logistic Regression tuned params: {logreg_search.best_params_} \n')
print(f'Random Forest tuned params: {rf_search.best_params_} \n')
print(f'XGBoost tuned params: {xgb_search.best_params_} \n')
print(f'SVC tuned params: {svc_search.best_params_} \n')
```

```
kNN tuned params: {'kneighborsclassifier__n_neighbors': 1}

Logistic Regression tuned params: {'logisticregression__C': 1, 'logisticregression__class_weight': None, 'logisticregression__penalt
y': 'l2', 'logisticregression__solver': 'liblinear'}

Random Forest tuned params: {'randomforestclassifier__class_weight': None, 'randomforestclassifier__max_depth': 5, 'randomforestclassi
fier__min_samples_split': 2, 'randomforestclassifier__n_estimators': 100}

XGBoost tuned params: {'xgbclassifier__learning_rate': 0.1, 'xgbclassifier__max_depth': 2, 'xgbclassifier__min_child_weight': 1, 'xgbc
lassifier__n_estimators': 100}

SVC tuned params: {'svc__C': 100, 'svc__class_weight': None, 'svc__gamma': 'scale', 'svc__kernel': 'rbf'}
```

Tuned Models

In [23]:
```
tuned_model_knn  = knn_search.best_estimator_
y_train_pred_knn = tuned_model_knn.predict(X_train)
y_pred_knn       = tuned_model_knn.predict(X_test)

tuned_model_lr   = logreg_search.best_estimator_
y_train_pred_lr  = tuned_model_lr.predict(X_train)
y_pred_lr        = tuned_model_lr.predict(X_test)

tuned_model_rf   = rf_search.best_estimator_
y_train_pred_rf  = tuned_model_rf.predict(X_train)
y_pred_rf        = tuned_model_rf.predict(X_test)

tuned_model_xgb  = xgb_search.best_estimator_
y_train_pred_xgb = tuned_model_xgb.predict(X_train)
y_pred_xgb       = tuned_model_xgb.predict(X_test)

tuned_model_svc  = svc_search.best_estimator_
y_train_pred_svc = tuned_model_svc.predict(X_train)
y_pred_svc       = tuned_model_svc.predict(X_test)
```

7 | Performance Metrics

Train vs Test Accuracies

In [24]:
```
train_test_dict = {'kNN':[y_train_pred_knn, y_pred_knn], 'Logistic Regression':[y_train_pred_lr, y_pred_lr],
                   'Random Forest':[y_train_pred_rf, y_pred_rf], 'XG Boost':[y_train_pred_xgb, y_pred_xgb],
                   'SVC':[y_train_pred_svc, y_pred_svc]}

for key, preds in train_test_dict.items():
    print(f"{key:<20} train:{round(accuracy_score(y_train, preds[0]),3):<5} test:{round(accuracy_score(y_test, preds[1]),3)}")
```

```
kNN                  train:1.0   test:0.961
Logistic Regression  train:0.925 test:0.843
Random Forest        train:0.99  test:0.902
XG Boost             train:0.97  test:0.863
SVC                  train:0.995 test:0.863
```

> The train and test scores look good.
>
> It appears KNN is the best model.

Stratified Cross Validated Accuracies

In [25]:
```
strat_kfold     = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

val_scores_dict = {'kNN':tuned_model_knn, 'Logistic Regression':tuned_model_lr, 'Random Forest':tuned_model_rf,
                   'XG Boost':tuned_model_xgb, 'SVC':tuned_model_svc}


for key, model in val_scores_dict.items():
    val_scores     = cross_val_score(model, X,y, cv=strat_kfold)
    rounded_scores = [round(score, 3) for score in val_scores]
    print(f'{key:<20}: {rounded_scores}\n')
```

```
kNN                 : [0.922, 0.96, 0.84, 0.96, 0.96]

Logistic Regression : [0.863, 0.82, 0.86, 0.88, 0.94]

Random Forest       : [0.941, 0.9, 0.88, 0.88, 0.98]

XG Boost            : [0.961, 0.86, 0.86, 0.86, 0.96]

SVC                 : [0.863, 0.88, 0.9, 0.92, 0.98]
```

> The scores are fairly close together for each model, suggesting that the models are stable and consistent across different subsets of the data.

Classification Reports

In [26]:
```
classification_dict = {'kNN':y_pred_knn, 'Logistic Regression':y_pred_lr,
                'Random Forest': y_pred_rf, 'XG Boost': y_pred_xgb,
                'SVC': y_pred_svc}

for key, pred in classification_dict.items():
    print(f'{key}:\n {classification_report(y_test, pred)}')
    print('-'*60)
```

kNN:

	precision	recall	f1-score	support
0	0.95	0.95	0.95	19
1	0.97	0.97	0.97	32
accuracy			0.96	51
macro avg	0.96	0.96	0.96	51
weighted avg	0.96	0.96	0.96	51

Logistic Regression:

	precision	recall	f1-score	support
0	0.87	0.68	0.76	19
1	0.83	0.94	0.88	32
accuracy			0.84	51
macro avg	0.85	0.81	0.82	51
weighted avg	0.85	0.84	0.84	51

Random Forest:

	precision	recall	f1-score	support
0	0.94	0.79	0.86	19
1	0.89	0.97	0.93	32
accuracy			0.90	51
macro avg	0.91	0.88	0.89	51
weighted avg	0.91	0.90	0.90	51

XG Boost:

	precision	recall	f1-score	support
0	0.93	0.68	0.79	19
1	0.84	0.97	0.90	32
accuracy			0.86	51
macro avg	0.88	0.83	0.84	51
weighted avg	0.87	0.86	0.86	51

SVC:

	precision	recall	f1-score	support
0	0.93	0.68	0.79	19
1	0.84	0.97	0.90	32

```
    accuracy                           0.86        51
   macro avg       0.88      0.83      0.84        51
weighted avg       0.87      0.86      0.86        51

------------------------------------------------------------
```

> Still, kNN appears to be the best model.
>
> Looking at the chosen hyperparameters, the optimal k is k=1. This means it predicts the class of a test point based only on its nearest neighbor. This would usually be cause for concern, but in this case I think the data has a fairly low amount of noise and the feautures are very high quality as shown in the box plots earlier - making a point's nearest neighbor a good predictor.
>
> All of the models except logistic regression have a recall of 0.97 for the positive class. Correctly identifying patients with diabetes is the most important aspect of the model. I want to make that recall score even higher without having a false positive rate that is too high.

---

8 | Ensemble Model with Soft Voting

In [27]:
```
ensemble_model = VotingClassifier(
     estimators=[('knn', tuned_model_knn), ('rf', tuned_model_rf), ('svc', tuned_model_svc),
                 ('xgb', tuned_model_xgb)], voting='soft')

ensemble_model.fit(X_train, y_train)

pred_ensemble       = ensemble_model.predict(X_test)
pred_ensemble_train = ensemble_model.predict(X_train)
```

Train vs Test

In [28]:
```
print(f"train:{round(accuracy_score(y_train, pred_ensemble_train),3)}")
print(f"test:{round(accuracy_score(y_test, pred_ensemble),3)}")
```

```
train:1.0
test:0.922
```

Classification Report

In [29]:
```
print(classification_report(y_test, pred_ensemble))
```

```
              precision    recall  f1-score   support

           0       0.94      0.84      0.89        19
           1       0.91      0.97      0.94        32

    accuracy                           0.92        51
   macro avg       0.93      0.91      0.91        51
weighted avg       0.92      0.92      0.92        51
```

Stratified Cross Val Scores

In [30]:
```
cross_val_score(ensemble_model, X, y, cv=strat_kfold)
```

Out[30]:
```
array([0.94117647, 0.92      , 0.88      , 0.94      , 0.98      ])
```

9 | Further Comparisons of Random Forest, kNN and Ensemble Model

Standard Deviations of Cross Val Scores

In [31]:
```
knn_cross_vals      = [0.922, 0.96, 0.84, 0.96, 0.96]
rf_cross_vals       = [0.941, 0.9, 0.88, 0.88, 0.98]
ensemble_cross_vals = [0.941, 0.92, 0.88, 0.94, 0.98]

print(f'KNN_Std:      {round(np.std(knn_cross_vals),2)}')
print(f'RF_Std:       {round(np.std(rf_cross_vals),2)}')
print(f'Ensemble_Std: {round(np.std(ensemble_cross_vals),2)}')
```

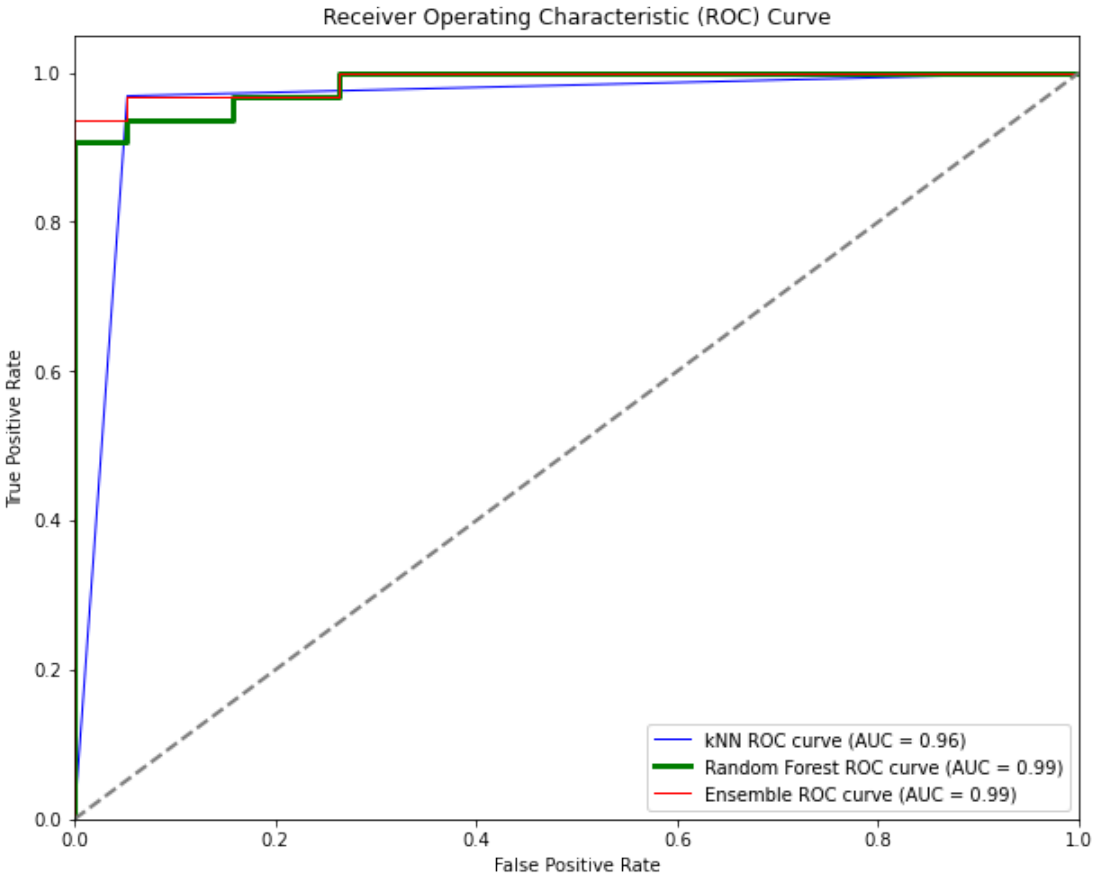
```
KNN_Std:      0.05
RF_Std:       0.04
Ensemble_Std: 0.03
```

The ensemble model has the lowest standard deviation regarding stratified cross val scores. This indicates it is the most stable across different subsets of the data. However these differences are small.

ROC Curves

In [32]:
```
y_pred_proba_knn      = tuned_model_knn.predict_proba(X_test)[:, 1]
y_pred_proba_rf       = tuned_model_rf.predict_proba(X_test)[:, 1]
y_pred_proba_ensemble = ensemble_model.predict_proba(X_test)[:, 1]
```

In [33]:
```
#
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_pred_proba_knn)
roc_auc_knn         = roc_auc_score(y_test, y_pred_proba_knn)

fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_proba_rf)
roc_auc_rf        = roc_auc_score(y_test, y_pred_proba_rf)

fpr_ensemble, tpr_ensemble, _ = roc_curve(y_test, y_pred_proba_ensemble)
roc_auc_ensemble              = roc_auc_score(y_test, y_pred_proba_ensemble)


plt.figure(figsize=(10, 8))
plt.plot(fpr_knn, tpr_knn, color='blue', lw=1, label=f'kNN ROC curve (AUC = {roc_auc_knn:.2f})')
plt.plot(fpr_rf, tpr_rf, color='green', lw=3, label=f'Random Forest ROC curve (AUC = {roc_auc_rf:.2f})')
plt.plot(fpr_ensemble, tpr_ensemble, color='red', lw=1, label=f'Ensemble ROC curve (AUC = {roc_auc_ensemble:.2f})')

plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")

plt.show()
```



> In terms of the max tpr-fpr, the best model appears to be the ensemble model with a fpr of 0% and a tpr of around 95%. The Ensemble and kNN models are tied for a classifier with a tpr around 96% and a fpr of around 5%.
>
> If a higher false positive rate (around 25%-28%) is tolerable in exchange for a tpr of 100%, then either Random Forest or the Ensemble model would be preferable.
>
> I will go with a model that has a higher fpr and a perfect tpr. As the base ensemble model performed better than the base random forest model, the ensemble model will be used.

---

10 | Optimal Threshold for the Ensemble Model

In [34]:
```
fpr_ensemble, tpr_ensemble, thresholds = roc_curve(y_test, y_pred_proba_ensemble)
roc_auc_ensemble                       = roc_auc_score(y_test, y_pred_proba_ensemble)
```

In [35]:
```
tpr_1_index             = np.where(tpr_ensemble == 1.0)[0][0]
optimal_threshold_tpr_1 = thresholds[tpr_1_index]

print(f"Threshold for TPR=1.0: {optimal_threshold_tpr_1}")
```

```
Threshold for TPR=1.0: 0.4253152519837502
```

In [36]:
```
y_pred_tpr_1 = (y_pred_proba_ensemble >= optimal_threshold_tpr_1).astype(int)
```

In [37]:
```
print(classification_report(y_test, y_pred_tpr_1))
```

```
              precision    recall  f1-score   support

           0       1.00      0.74      0.85        19
           1       0.86      1.00      0.93        32

    accuracy                           0.90        51
   macro avg       0.93      0.87      0.89        51
weighted avg       0.92      0.90      0.90        51
```

> These seem like really good results. The overall accuracy is 90% and the recall for the positive class is 100%.

Conditional Probabilities

Let P = Positive, N = Negative

P(P|Tested Positive) = precision of positive class = 0.86

P(N|Tested Positive) = 1 - precision of positive class = 0.14

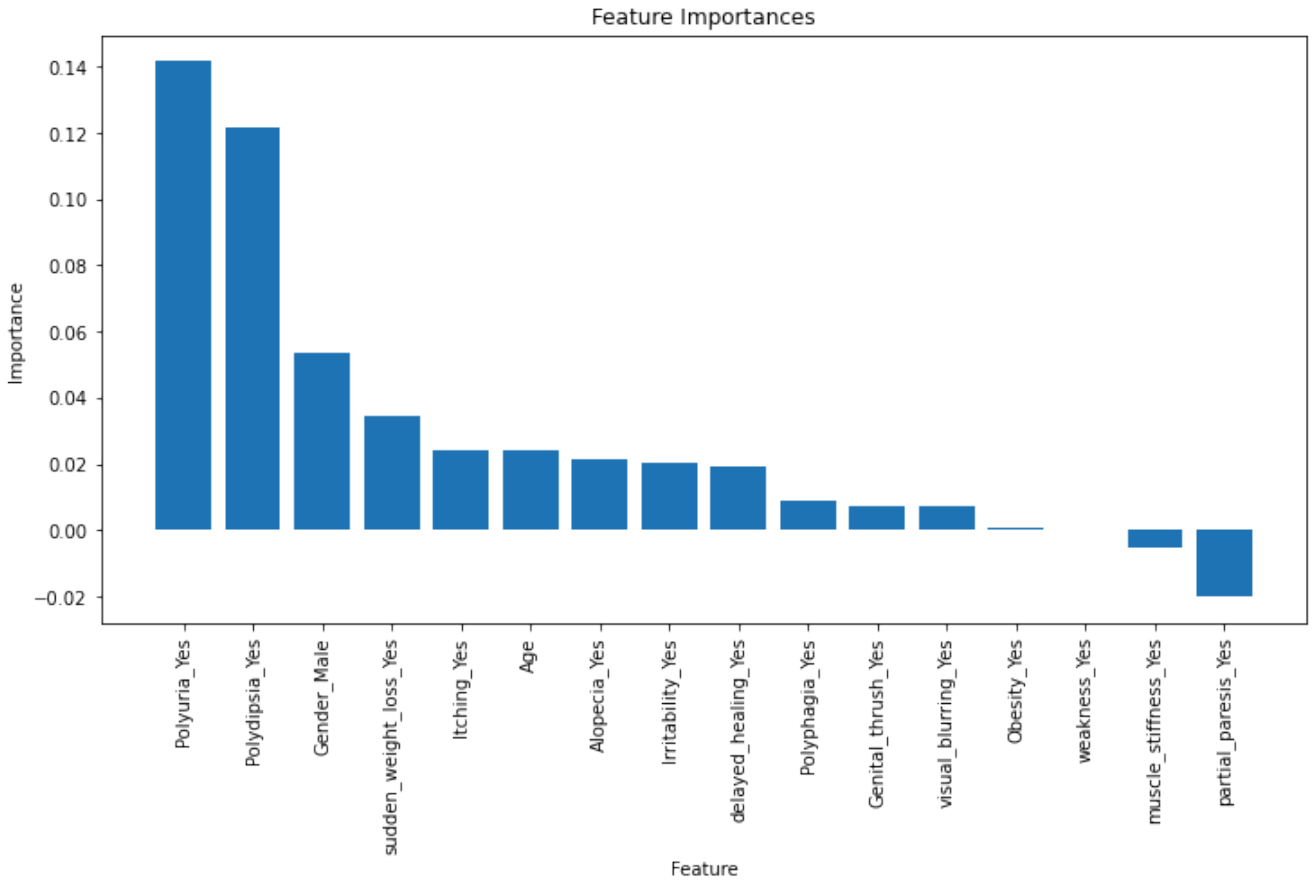
P(N|Tested Negative) = precision of negative class = 1.0

P(P|Tested Negative) = 1 - precision of negative class = 0.0

> As shown by the conditional probabilities, the probability of a patient being positive for diabetes given that they tested negative is 0% for this data. This is the number I wanted to minimize, so these results are good.

Approximated Feature Importance

In [38]:
```
perm_importance            = permutation_importance(ensemble_model, X_test, y_test, n_repeats=50, random_state=1)

feature_importances        = pd.DataFrame({'Feature': X.columns, 'Importance': perm_importance.importances_mean})
sorted_feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(12,6))
plt.bar(sorted_feature_importances['Feature'], sorted_feature_importances['Importance'])
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importances')
plt.xticks(rotation=90)
plt.show()
```



> The most important features appear to be Polyuria, Polydispsia, Gender and so on.
>
> I believe muscle stiffness and partial paresis having negative importance means no importance here. A correlation heatmap did not indicate severe multicolinearity with these variables.

---

11 | Conclusion

In conclusion, the model chosen for this project was an ensemble model with soft voting made up of kNN, Random Forest, SVC and XGBoost models. This model was chosen for having a perfect recall score for the positive class while also having what seems like a reasonable false positive rate.

Random forest could have been chosen for the same reasons. As shown in the ROC curve plot, random forest and the ensemble model overlapped at this point of interest. However the ensemble model appeared to be better based on 1) the initial classification reports and 2) the standard deviations of the stratified cross val scores. It is assumed that the added complexity of the ensemble model is acceptable.

One noteable thing about the population in this dataset is that about 69% of the patients were positive for diabetes and about 31% negative. These percentages would not match the general population to my knowledge. Also, the data contains about 64% males and 36% females, so it is unbalanced there. Furthermore, a much higher percentage of the females in popuation tested positive for diabetes as compared to the males. I don't think this ratio is representative. These are things that would need to be considered regarding this model.