In [577]:
```
import pandas as pd
```

In [578]:
```
df = pd.read_csv("C:/Users/nneam/OneDrive/Documents/540Assignments/Diabetes_Classification.csv")
```

In [579]:
```
df.head()
```

Out[579]:

	id	Cholesterol	Glucose	HDL Chol	Chol/HDL ratio	Age	Gender	Height	Weight	BMI	Systolic BP	Diastolic BP	waist	hip	Waist/hip ratio	Diabetes	Unnamed: 16
0	1	193	77	49	3.9	19	female	61	119	22.5	118	70	32	38	0.84	No diabetes	6.0
1	2	146	79	41	3.6	19	female	60	135	26.4	108	58	33	40	0.83	No diabetes	NaN
2	3	217	75	54	4.0	20	female	67	187	29.3	110	72	40	45	0.89	No diabetes	NaN
3	4	226	97	70	3.2	20	female	64	114	19.6	122	64	31	39	0.79	No diabetes	NaN
4	5	164	91	67	2.4	20	female	70	141	20.2	122	86	32	39	0.82	No diabetes	NaN

In [580]:
```
#Dropping unneccesary columns
df2 = df.drop(columns=['Unnamed: 16', 'Unnamed: 17'])
```

In [581]:
```
#Replcing diabetes and gender column value for model
df2['Diabetes'] = df2['Diabetes'].replace(['No diabetes','Diabetes'],['0','1'])
df2['Gender'] = df2['Gender'].replace(['female','male'],['0','1'])
df2.head()
```

Out[581]:

	id	Cholesterol	Glucose	HDL Chol	Chol/HDL ratio	Age	Gender	Height	Weight	BMI	Systolic BP	Diastolic BP	waist	hip	Waist/hip ratio	Diabetes
0	1	193	77	49	3.9	19	0	61	119	22.5	118	70	32	38	0.84	0
1	2	146	79	41	3.6	19	0	60	135	26.4	108	58	33	40	0.83	0
2	3	217	75	54	4.0	20	0	67	187	29.3	110	72	40	45	0.89	0
3	4	226	97	70	3.2	20	0	64	114	19.6	122	64	31	39	0.79	0
4	5	164	91	67	2.4	20	0	70	141	20.2	122	86	32	39	0.82	0

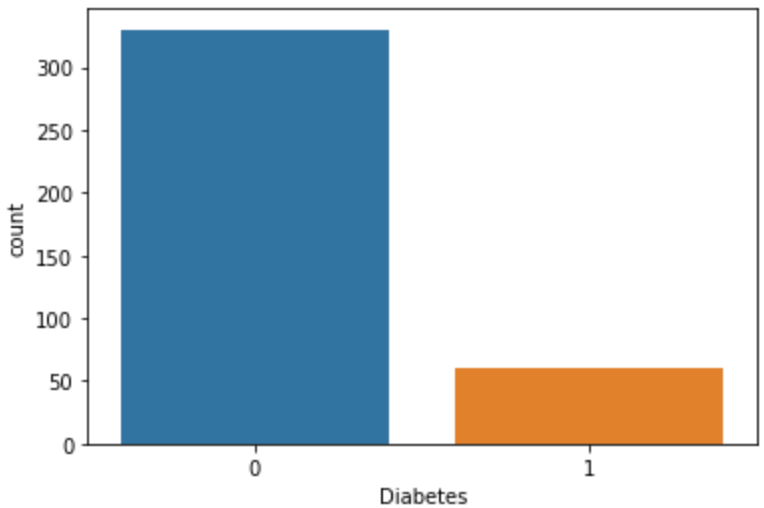
In [582]:
```
#Loading Libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

EDA

In [583]:
```
#Counting Diabetes Coulumn Values
df2['Diabetes'].value_counts()
```

Out[583]:
```
0    330
1     60
Name: Diabetes, dtype: int64
```

In [584]:
```
#Visualizing
sns.countplot(x='Diabetes', data = df2)
plt.show()
```



In [585]:

```
# Analyzing data by gender
df2.groupby('Gender').mean()
```

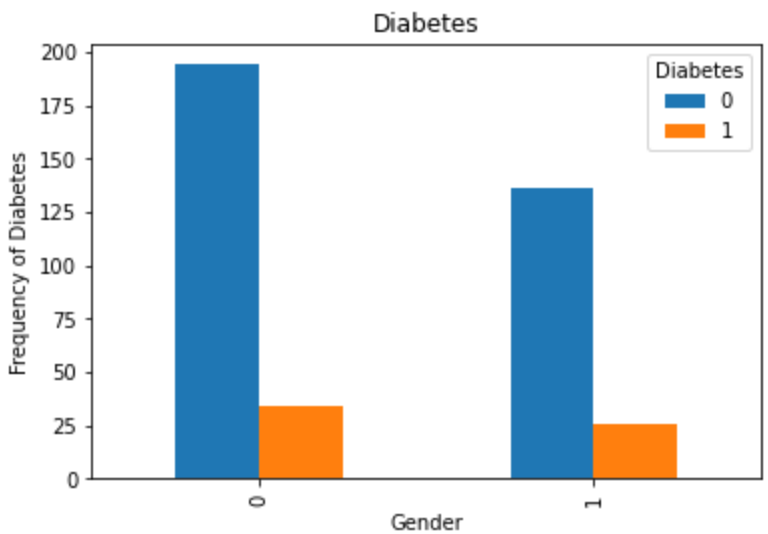
Out[585]:

Gender	id	Cholesterol	Glucose	HDL Chol	Chol/HDL ratio	Age	Height	Weight	BMI	Systolic BP	Diastolic BP	wais
0	186.942982	208.364035	103.109649	51.842105	4.374123	45.609649	63.714912	174.276316	30.188158	136.451754	82.482456	38.09210
1	207.543210	205.635802	113.290123	48.049383	4.736420	48.413580	69.098765	181.814815	26.787654	138.092593	84.425926	37.55555

In [586]:

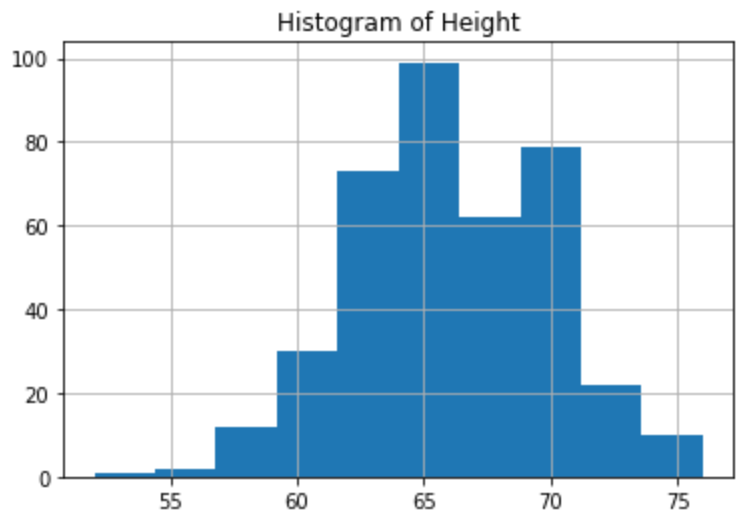
```
#Visualizing
%matplotlib inline
pd.crosstab(df2.Gender,df2.Diabetes).plot(kind='bar')
plt.title('Diabetes')
plt.xlabel('Gender')
plt.ylabel('Frequency of Diabetes')
```

Out[586]: Text(0, 0.5, 'Frequency of Diabetes')



In [587]:
```
#Histogram for Height
df2.Height.hist()
plt.title('Histogram of Height')
```

Out[587]: Text(0.5, 1.0, 'Histogram of Height')



Standard Logistic Regression Model

In [588]:

```
#Analyzing data types
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 390 entries, 0 to 389
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               390 non-null    int64
 1   Cholesterol      390 non-null    int64
 2   Glucose          390 non-null    int64
 3   HDL Chol         390 non-null    int64
 4   Chol/HDL ratio   390 non-null    float64
 5   Age              390 non-null    int64
 6   Gender           390 non-null    object
 7   Height           390 non-null    int64
 8   Weight           390 non-null    int64
 9   BMI              390 non-null    float64
 10  Systolic BP      390 non-null    int64
 11  Diastolic BP     390 non-null    int64
 12  waist            390 non-null    int64
 13  hip              390 non-null    int64
 14  Waist/hip ratio  390 non-null    float64
 15  Diabetes         390 non-null    object
dtypes: float64(3), int64(11), object(2)
memory usage: 48.9+ KB
```

In [589]:

```
#Changing value type
df2['Diabetes'] = df2.Diabetes.astype(int)
df2['Gender'] = df2.Gender.astype(int)
```

In [590]:

```
#Verifying change
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 390 entries, 0 to 389
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               390 non-null    int64
 1   Cholesterol      390 non-null    int64
 2   Glucose          390 non-null    int64
 3   HDL Chol         390 non-null    int64
 4   Chol/HDL ratio   390 non-null    float64
 5   Age              390 non-null    int64
 6   Gender           390 non-null    int32
 7   Height           390 non-null    int64
 8   Weight           390 non-null    int64
 9   BMI              390 non-null    float64
 10  Systolic BP      390 non-null    int64
 11  Diastolic BP     390 non-null    int64
 12  waist            390 non-null    int64
 13  hip              390 non-null    int64
 14  Waist/hip ratio  390 non-null    float64
 15  Diabetes         390 non-null    int32
dtypes: float64(3), int32(2), int64(11)
memory usage: 45.8 KB
```

In [591]:

```
#Creating the Test, Train, and Split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df2.drop('Diabetes',axis=1), df2['Diabetes'], test_size=0.20, rand
```

In [592]:

```
#Training Model
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression(solver='liblinear', max_iter=200)
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
```

In [593]:

```
#Evaluating Model
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        63
           1       0.82      0.60      0.69        15

    accuracy                           0.90        78
   macro avg       0.86      0.78      0.82        78
weighted avg       0.89      0.90      0.89        78
```

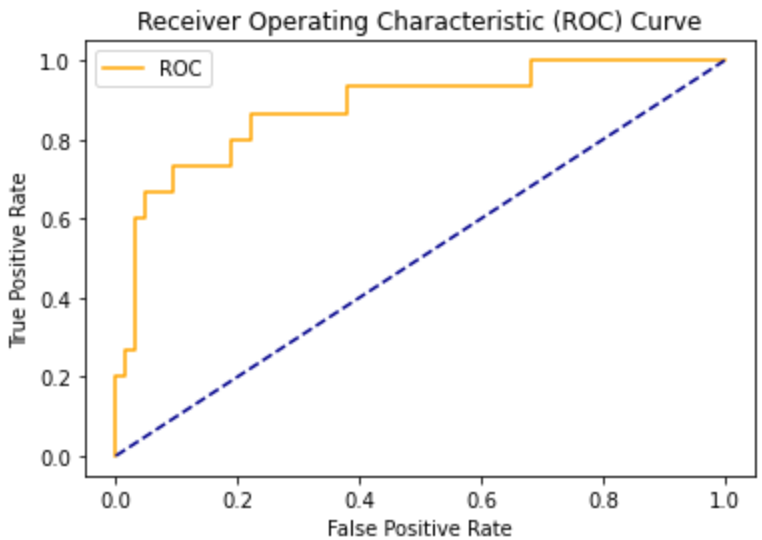
In [594]:

```python
# Roc Curve
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

probs = model.predict_proba(X_test)
probs = probs[:, 1]
auc = roc_auc_score(y_test, probs)
print('AUC: %.2f' % auc)
fpr, tpr, thresholds = roc_curve(y_test, probs)
plot_roc_curve(fpr, tpr)
```

```
AUC: 0.88
```



In [644]:

```
# This portion we will test the regulation strength and compare results

#Training Model
logmodel = LogisticRegression(solver='liblinear', max_iter=200, C = 100.0)
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
#Evaluating Model
print(classification_report(y_test,predictions))

# We can see adjusting this parameter didn't help our results next I'll
# add a penalty parameter
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95        64
           1       0.90      0.64      0.75        14

    accuracy                           0.92        78
   macro avg       0.91      0.81      0.85        78
weighted avg       0.92      0.92      0.92        78
```

In [645]:

```
# L2 penalty test


#Training Model
logmodel = LogisticRegression(solver='liblinear', max_iter=200, penalty='l1')
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
#Evaluating Model
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95        64
           1       0.90      0.64      0.75        14

    accuracy                           0.92        78
   macro avg       0.91      0.81      0.85        78
weighted avg       0.92      0.92      0.92        78
```

In [646]:

```
# Li penalty test


#Training Model
logmodel = LogisticRegression(solver='liblinear', max_iter=200, penalty='l2')
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
#Evaluating Model
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.95        64
           1       0.89      0.57      0.70        14

    accuracy                           0.91        78
   macro avg       0.90      0.78      0.82        78
weighted avg       0.91      0.91      0.90        78
```

Support vector

Adjusting the C and Penalty parameters did not improve model, will try adding a support vector next

In [598]:

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

In [599]:

```
#Divide Data
X = df2.drop('Diabetes', axis=1)
y = df2['Diabetes']
```

In [600]:

```
#Assign test and training data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state=200)
```

In [601]:

```
#Set Kernel for linear
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
#Make Predictions and Evaluate
y_pred = svclassifier.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        63
           1       0.82      0.60      0.69        15

    accuracy                           0.90        78
   macro avg       0.86      0.78      0.82        78
weighted avg       0.89      0.90      0.89        78
```

In [602]:

```
#Set Kernel for poly
# Testing the regulazationion parameter negatively impacted the model

#Set Kernel
svclassifier = SVC(kernel='poly')
svclassifier.fit(X_train, y_train)
#Make Predictions and Evaluate
y_pred = svclassifier.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        63
           1       0.82      0.60      0.69        15

    accuracy                           0.90        78
   macro avg       0.86      0.78      0.82        78
weighted avg       0.89      0.90      0.89        78
```

In [603]:

```
#Set Kernel for rbf


#Set Kernel
svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)
#Make Predictions and Evaluate
y_pred = svclassifier.predict(X_test)
#classification report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        63
           1       0.82      0.60      0.69        15

    accuracy                           0.90        78
   macro avg       0.86      0.78      0.82        78
weighted avg       0.89      0.90      0.89        78
```

Grid Search

In [604]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

In [605]:

```
#Split data
X = df2.drop('Diabetes', axis=1)
y = df2['Diabetes']
```

In [606]:

```
#Assign Training and Test
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=200)
```

In [607]:

```
#Create Grid Search Model
grid={"C":np.logspace(-3,3,7), "penalty":["l2"]}
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,grid,cv=10)
logreg_cv.fit(X_train,y_train)
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
C:\Users\nneam\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocess
ing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stabl
e/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
C:\Users\nneam\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocess
ing.html)
Please also refer to the documentation for alternative solver options:
```

In [608]:

```
#Classification Report
print(classification_report(logreg_cv.best_estimator_.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        67
           1       0.60      0.82      0.69        11

    accuracy                           0.90        78
   macro avg       0.78      0.86      0.82        78
weighted avg       0.92      0.90      0.90        78
```

Random Forrest

In [609]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

In [610]:
```
X = df2.drop('Diabetes', axis=1)
y = df2['Diabetes']
```

In [611]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

In [612]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [640]:
```
#20 trees offer best results
regressor = RandomForestRegressor(n_estimators=20, random_state=0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
print(classification_report(y_test,y_pred.round()))
print(accuracy_score(y_test, y_pred.round()))
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96        64
           1       0.91      0.71      0.80        14

    accuracy                           0.94        78
   macro avg       0.92      0.85      0.88        78
weighted avg       0.93      0.94      0.93        78

0.9358974358974359
```

In [ ]: