Exercise 1

> Data Handling exercise

This exercise is for those who are already familiar with python

A Visual Exploration and Statistical Analysis of a Diabetes Dataset using Python

This Dataset is Freely Available

Overview:

The data was collected and made available by the "National Institute of Diabetes and Digestive and Kidney Diseases" as part of the Pima Indians Diabetes Database.

`Diabetes.csv` is available [from Kaggle (https://www.kaggle.com/uciml/pima-indians-diabetes-database)](https://www.kaggle.com/uciml/pima-indians-diabetes-database).
+++++++++++++++++++++++++++++++++++++

The following features are present in the dataset:

- **Pregnancies:** Number of times pregnant
- **Glucose:** Plasma glucose concentration over 2 hours in an oral glucose tolerance test
- **BloodPressure:** Diastolic blood pressure (mm Hg)
- **SkinThickness:** Triceps skin fold thickness (mm)
- **Insulin:** 2-Hour serum insulin (mu U/ml)
- **BMI:** Body mass index (weight in kg/(height in m)2)
- **DiabetesPedigreeFunction:** Diabetes pedigree function (a function which scores likelihood of diabetes based on family history)
- **Age:** Age (years)
- **Outcome:** Class variable (0 if non-diabetic, 1 if diabetic)

Scenario: Imagine you have collected this data and wish to analyse it

In Moodle, you are going to work with a data named "Diabetess.csv".

It is the same Diabetes dataset but saved as "Diabetess.csv" for this exercise.

A. Here are the expected things to do in this exercise

> 1. Import the necessary/required libraries.
> 2. Load the data (Diabetes.csv)
> 3. Show the information about the data.
> 4. Describe the data

Points allocated for this step: (0.5 point)

In [ ]:
```
pip install seaborn
```

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
#Read the input data to dataframe.
df = pd.read_csv('Diabetess.csv')
```

In [40]:
```
#See how data looks.
df.head()
```

Out[40]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72	35	206.846154	33.6	0.627	50.0	1
1	1	85.0	66	29	130.287879	26.6	0.351	31.0	0
2	8	183.0	64	0	206.846154	23.3	0.672	32.0	1
3	1	89.0	66	23	94.000000	28.1	0.167	21.0	0
4	0	137.0	40	35	168.000000	43.1	2.288	33.0	1

In [ ]:
```
#Check for null values.
df.isnull()
```

In [ ]:
```
#list all the columns and data types. Check if features/label are missing data.
df.info()
```

Glucose, Insulin and BMI missing data. Insuling almost 50% missing data. Replace with values or remove after analysis.

In [ ]:
```
#Check data ranges and potential outliers. How is ditribution.
df.describe()
```

B. Handle the missing data

Missing and Zero Values

- It is clear the data has missing and zero values
- For example, we can see that SkinThickness = 0 in the third row
- And we can also see some 'NaN' values
- Sometimes missing values are represented by a '?'
- Get more info and decide on how to handle these issues as instructed below:

5. If the data uses '?' for missing values then we can replace them with a NaN.

Insulin has a large number of missing values .. so we can drop that column

6. Drop all rows that contain missing values.
7. Drop all rows that contain missing values?
8. Show the shape of the data.
9. Describe the data.
10. Show the information of the cleaned data.

Points allocated for this step: (0.5 point)

The Mean and Median

- **The mean** is the simple mathematical average of a list of two or more numbers.
- **The median** is the middle number in a sorted, ascending or descending, list of numbers and can be more descriptive of that data set than the average.

[https://www.investopedia.com/terms-beginning-with-m-4769363 (https://www.investopedia.com/terms-beginning-with-m-4769363)](https://www.investopedia.com/terms-beginning-with-m-4769363)

Zero Values that Don't Make Sense

In case you want would like to replace the missing values by the mean or median value

Here is an example of the code below

In [3]:

```
### Replace missing values in each column with the mean or median of that column.
#data = df.fillna(df.mean())
#data = df.fillna(df.median())
#data = pd.concat([df.ffill(), df.bfill()]).groupby(level=0).median()
#Fill in missing insuling data according to outcome type insulin mean.
tmp = df[df['Insulin'].notnull()]
tmp = tmp[['Insulin', 'Outcome']].groupby(['Outcome'])[['Insulin']].mean().reset_index()
df.loc[(df['Outcome'] == 0 ) & (df['Insulin'].isnull()), 'Insulin'] = tmp.iloc[0]['Insulin']
df.loc[(df['Outcome'] == 1 ) & (df['Insulin'].isnull()), 'Insulin'] = tmp.iloc[1]['Insulin']
data=df.fillna(df.mean())
```

In our exercise, remove all the rows with zero with the exception of the first and last columns.

Here is an example of the code to that:

In [4]:

```
## get data where non of the columns has 0 value (except the first and last columns). Check t
clean_data = data[~(data[data.columns[1:-1]] == 0).any(axis=1)]
clean_data.shape
```

Out[4]: (539, 9)

In [ ]:

```
#Have a look on cleaned data.
clean_data.head(10)
```

In [ ]:
```
#All the features/attributes and label have same count!! No missing or null values found.
clean_data.info()
```

Some Summary Information

In [ ]:
```
%%HTML
<style type="text/css">
table.dataframe td, table.dataframe th {
    border: 1px  black solid !important;
  color: black !important;
}
</style>
```

In [ ]:
```
#Check the data ranges once more.
clean_data.describe()
```

In [ ]:
```
## check the mean of values depending on their category (i.e. 0 or 1)
clean_data.groupby('Outcome').mean()
```

In [ ]:
```
# the difference between the mean and median is a good indicator of how much skewed your data
clean_data.groupby('Outcome').agg(['mean','median'])
```

Insulin mean and median delta is elevated. This is for cases with diabetes. NaN mean/meadian replacement might not be correct way to do it. Further analysis shows it is elevated even before NaN replacement. One option is to drop 'Insulin' column from final data.

2- Useful and Informative Plots

Histogram Plots

Plot the histogram plots of each variable

> Allocated points (1 point)

In [ ]:
```
#Histogram plots
plt.figure(figsize=(16,12))
for i in range(1, 10):
    plt.subplot(3, 3, i)
    col = clean_data.columns[i-1]
    plt.hist(clean_data[col], bins=30)
    plt.title(col)
```

Scatter Matrix

- This one is a useful one liner ... but note that it only works with numeric data
- If you want to include categorical data in there you should convert the categories into numeric labels

> Allocated points (1 point)

In [ ]:
```
#Scatter matrix, Outcome used for coloring scheme. Blue - No, Orange - yes
sns.pairplot(clean_data, hue="Outcome")
```

It looks Glucose have stronger correlation with outcome and other features. Distribution for non-diabetes cases show normal distribution. Positive cases show some non-normal distribution. Will this require normalization?

In [ ]:

```
#Let's have closer look on some of the data.
fig, ax = plt.subplots()
x=clean_data['BMI']
y=clean_data['Glucose']
ax.scatter(x, y, c=clean_data['Outcome'], s=clean_data['Glucose']/2)
ax.legend(x)
ax.grid(True)
plt.show()
```

In [ ]:

```
#3D plotting
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
x = clean_data['Age']
y = clean_data['BMI']
z = clean_data['Glucose']
ax.scatter(x, y, z,c=clean_data['Outcome'])
ax.set_xlabel("Age")
ax.set_ylabel("BMI")
ax.set_zlabel("Glucose")

plt.show()
```

In [ ]:

```
#Heatmap for feature and label correlation.
corr = clean_data.corr()
sns.heatmap(corr, annot=True)
```

Conclusion

You can submit the exercise by submitting the Jupyter notebook file (ipynb) or as pdf

> The main aim of this exercise is to explore the data and handle the possibility of missing data (rows or columns)

Notes from histograms and scatter matrix.###

Insulin Null data replaced with mean or median will create high spike for this category as almost half of the data is not available. This potentially can skew the outcome. Look for other option to replace missing value. bfill/ffill used. Also Insulin-Glucose there is a correlation according to heatmap. Other option to fill missing insulin values taking values matching Glucose. Clucose level has high correlation to outcome. Age is not evenly balanced as most samples are <50 years. Check accuracy of model once trained. Non-diabetics are double amount in 'Outcome' category. Not totally balanced data but still workable. Model training and validation will show if any further data wrangling is needed. Scaling and normalization also options.

The same data will be used for exercise 2.

Exercise 2 continues from here.

Well Done!

Exercise 2 Classification task

In [5]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, accuracy_score
```

In [190]:

```
features = clean_data.drop(columns='Outcome')
#labels = np.array(clean_data['Outcome'])
labels = clean_data['Outcome']
```

In [191]:

```
#Scaling the data. Logistic regression is not much affcted if no scaling done. NN model big i
#I believe it is because weighting inside the network layers requires scaled input data to re
st = StandardScaler()
features_sca = st.fit_transform(features)
features_sca.shape
```

Out[191]:

```
(539, 8)
```

A: Logistic Regression Model

In [192]:

```
#Split the data.
X_train, X_test, y_train, y_test = train_test_split(features_sca, labels, test_size=0.4, ranc
```

In [193]:

```
#'class_weight' added and tried as number of diabetics is much less than non-diabetics in giv
#This improves diabetic recal rate from mid 50% to >70%. But overall model accuracy decreases
#l_clf = LogisticRegression(solver='liblinear', C=10.0, random_state=0, class_weight='balance
l_clf = LogisticRegression(random_state = 6,
                           solver = 'liblinear',
                           #class_weight='balanced',
                           multi_class='ovr',
                           penalty='l2')
```

In [194]:

```
#Fit the model
l_clf.fit(X_train, y_train)
```

Out[194]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='ovr', n_jobs=None, penalty='l2', random_state=6,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [195]:

```
#Create training KPIs
p_pred = l_clf.predict_proba(X_train)
y_pred = l_clf.predict(X_train)
score_ = l_clf.score(X_train, y_train)
conf_m = confusion_matrix(y_train, y_pred)
report = classification_report(y_train, y_pred)
```

In [ ]:

```
#Create test KPIs
p_pred = l_clf.predict_proba(X_test)
y_pred = l_clf.predict(X_test)
score_ = l_clf.score(X_test, y_test)
conf_m = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

In [196]:

```
#Function for confusion matrix creation.
def accuracy(confusion_matrix):
   diagonal_sum = confusion_matrix.trace()
   sum_of_all_elements = confusion_matrix.sum()
   return diagonal_sum / sum_of_all_elements
```

In [197]:

```
#Print accuracy using confusion matrix.
print('Confusion matrix:\n',conf_m)
print("Accuracy of LogisticRegression : ", accuracy(conf_m))
```

```
Confusion matrix:
 [[194  23]
 [ 46  60]]
Accuracy of LogisticRegression :  0.7863777089783281
```

In [ ]:

```
#Print complete KPI report.
print('report:', report, sep='\n')
```

In [ ]:

```
#ROC curve.
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

logit_roc_auc = roc_auc_score(y_train, l_clf.predict(X_train))
fpr, tpr, thresholds = roc_curve(y_train, l_clf.predict_proba(X_train)[:,1])

plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
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')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```

A: Neural Network

In [80]:

```
from sklearn.neural_network import MLPClassifier
```

In [ ]:

```
#Create NN network layout.
clf = MLPClassifier(solver='adam', #adam
                    #alpha=1e-4,
                    hidden_layer_sizes=(150,100,50), #150 best. 80 with lbfgs
                    random_state=24,
                    batch_size=8,
                    verbose=False,
                    early_stopping=True,
                    activation='relu',
                    learning_rate='constant',
                    learning_rate_init=0.005,
                    max_iter=300
                    )
```

In [ ]:

```
#Fit the model.
clf.fit(X_train, y_train)
```

In [ ]:

```
#Model training performance.
print("MLP training accuracy:",clf.score(X_train, y_train))
```

In [ ]:

```
#Model prediction performance.
print("MLP test accuracy:",clf.score(X_test, y_test))
```

In [ ]:

```
#Check what is probability of each test prediction.
clf.predict_proba(X_test)
```

In [ ]:

```
#NN model prediction.
y_pred = clf.predict(X_test)
cm = confusion_matrix(y_pred, y_test)
#print("Accuracy of MLPClassifier : ", accuracy(cm))
```

In [ ]:

```
#Check delta between prediction and real label.
dat1 = pd.DataFrame(y_test)
dat1['Prediction'] = y_pred
dat1.head()
```

In [ ]:

```
#Loss graph With more paramter optimization loss can be even lower.
plt.plot(clf.loss_curve_)
plt.title("Validtion Loss")
plt.xlabel("epoch")
plt.ylabel("Loss")
```

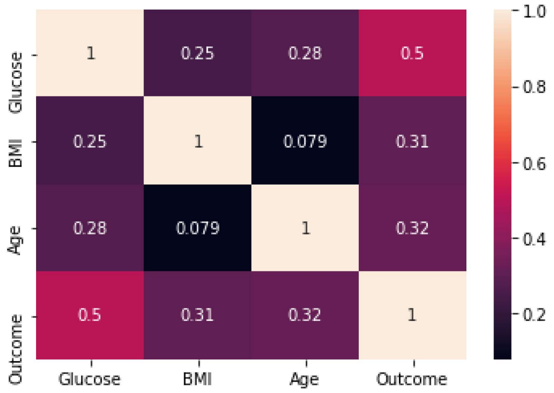
In [ ]:
```
#Show model parameters.
clf.get_params(deep=True)
```

B: Feature selection. Only BMI, glucose and age selected.

In [205]:
```
#Feature selection, drop features, only use glucose, BMI and age..

features_sel = clean_data.drop(columns=['Pregnancies',
                                    'BloodPressure',
                                    'SkinThickness',
                                    'Insulin',
                                    'DiabetesPedigreeFunction'])
```

In [206]:
```
#Check correlation of selected features with label.Glucose has the highest correlation with C
sns.heatmap(features_sel.corr(),annot=True)
plt.show()
```



In [183]:
```
st = StandardScaler()
features_sca2 = st.fit_transform(features_sel.drop(columns=['Outcome']))
features_sca2.shape
```

Out[183]: (539, 3)

In [184]:
```
#Split the data. Instructions has keep test data size 0.001.That would leave no data for test
#less than 1000 samples. I kept same 40% as a size.
X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(features_sca2,
                                                                    labels,
                                                                    test_size=0.4,
                                                                    random_state=24)#0.4 good
```

B: Logistic Regression Model

In [185]:

```
#Logistic regression model. Using only three features. Class wirght also tested as diabetic o
#35% of all.
l_clf = LogisticRegression(random_state = 6,
                           solver = 'liblinear',
                           #class_weight='balanced',
                           multi_class='ovr',
                           penalty='l2')
#Fit the model
l_clf.fit(X_train_sub, y_train_sub)
#Create training KPIs
p_pred = l_clf.predict_proba(X_train_sub)
y_pred = l_clf.predict(X_train_sub)
score_ = l_clf.score(X_train_sub, y_train_sub)
conf_m = confusion_matrix(y_train_sub, y_pred)
report = classification_report(y_train_sub, y_pred)
y_pred.shape
#Confusion matrix.
print('Confusion matrix:\n',conf_m)
#Accuracy
print("Training accuracy of MLPClassifier : ", accuracy(conf_m))
```

```
Confusion matrix:
 [[191  26]
 [ 47  59]]
Training accuracy of MLPClassifier :  0.7739938080495357
```

In [21]:

```
#Print complete KPI report.
print('report:', report, sep='\n')
```

```
report:
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       217
           1       0.69      0.56      0.62       106

    accuracy                           0.77       323
   macro avg       0.75      0.72      0.73       323
weighted avg       0.77      0.77      0.77       323
```

Observation: Not much delta on result after feature selection. 77.4% vs.78.6%.

B: NN Model

In [87]:

```
#Create NN network layout.
clf_sub = MLPClassifier(solver='adam',
                     #alpha=1e-4,
                     hidden_layer_sizes=(150,100,50), #150 best. 80 with lbfgs
                     random_state=24,
                     batch_size=8,
                     verbose=False,
                     early_stopping=True,
                     activation='relu',
                     learning_rate='constant',
                     learning_rate_init=0.005,
                     max_iter=300
                     )
```

In [88]:

```
#Fit the NN model with new data.
clf_sub.fit(X_train_sub, y_train_sub)
```

Out[88]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size=8, beta_1=0.9,
              beta_2=0.999, early_stopping=True, epsilon=1e-08,
              hidden_layer_sizes=(150, 100, 50), learning_rate='constant',
              learning_rate_init=0.005, max_fun=15000, max_iter=300,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=24, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

In [89]:
```
#NN model training performance.
print("MLP training accuracy:",clf_sub.score(X_train_sub, y_train_sub))
```

```
MLP training accuracy: 0.7801857585139319
```

In [90]:
```
#NN model test performance.
print("MLP test accuracy:",clf_sub.score(X_test_sub, y_test_sub))
```

```
MLP test accuracy: 0.8009259259259259
```

In [ ]:
```
#Loss graph. For last 4 epoch loss is already flatten.
plt.plot(clf_sub.loss_curve_)
plt.title("Validtion Loss")
plt.xlabel("epoch")
plt.ylabel("Loss")
```

Observation: NN network training accuracy went down after feature selection. 78.0% vs 87.6%. It seem NN is able to find more information from the excluded data.

C: Assuming a patient just has just walked into a clinic. The clinician wants to predict his/her outcome.

In [ ]:
```
#The patients' age is 35 years with a glucose level of 110 and BMI of 35.
patient1 = pd.DataFrame({'Glucose':[110], 'BMI':[35], 'Age':[35]})
patient1_scaled = st.fit_transform(patient1)
```

In [ ]:
```
#Logistic regression patient outcome prediction.
y_pred_prob = l_clf.predict_proba(patient1_scaled)
y_pred = l_clf.predict(patient1_scaled)
print("Outcome =", y_pred, "LRM Probability =",y_pred_prob)
```

LR model prediction is '0', non-diabetic. Probability is 71.7%.

In [ ]:
```
#NN patient outcome prediction.
y_pred_prob = clf_sub.predict_proba(patient1_scaled)
y_pred = clf_sub.predict(patient1_scaled)
print("Outcome =", y_pred, "MLPC Probability =",y_pred_prob)
```

Just for curiocity also NN was tested. NN model prediction is '0', non-diabetic. Probability is 57%. Logistic regression predicts outcome with higher confidence.

Well done...Submit your Python file as pdf file or as a python file.

Exercise 3 Feature Importance

Only BMI, Age and Glucose

In [186]:
```
#Logistic regression model feature importance / coefficiency. Only three features selected.
importance = l_clf.coef_
results = [item for elem in importance for item in elem]
print('Feature coeffresults',results)
```

```
Feature coeffresults [0.9263789724915099, 0.585013633169982, 0.4669880002578407]
```

In [187]:
```
#STD
print(np.std(X_train_sub, 0)*l_clf.coef_)
```

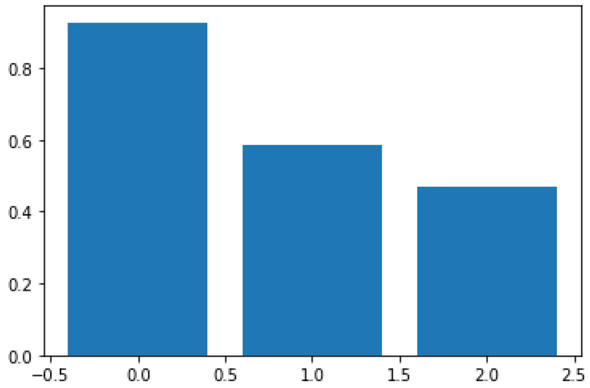
```
[[0.931863   0.5801206  0.46659567]]
```

In [180]:

```python
# Plot feature importance
for i,v in enumerate(results):
    print('Feature: %0d, Score: %.5f, %s' % (i,v,features_sel.columns[i]))
plt.bar([x for x in range(len(results))], results)

plt.show()
```

```
Feature: 0, Score: 0.92638, Glucose
Feature: 1, Score: 0.58501, BMI
Feature: 2, Score: 0.46699, Age
```



In [181]:

```python
#Pick only Glucose for model fit.
features_sel = features_sel.drop(columns=['BMI','Age','Outcome'])
X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(features_sel,
                                                                    labels,
                                                                    test_size=0.4,
                                                                    random_state=24)
#Fit the model
l_clf.fit(X_train_sub, y_train_sub)
y_pred = l_clf.predict(X_train_sub)
conf_m = confusion_matrix(y_train_sub, y_pred)
#Accuracy
print("Training accuracy of MLPClassifier : ", accuracy(conf_m))
```

```
Training accuracy of MLPClassifier :  0.7523219814241486
```

Accuracy reduced from 78.6% to 75.2%. Glucose only has high prediction power.

In [188]:

```python
#Remove Glucose from model fit.Use BMI and age only.
features_sel = features_sel.drop(columns=['Glucose','Outcome'])
X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(features_sel,
                                                                    labels,
                                                                    test_size=0.4,
                                                                    random_state=24)
#Fit the model
l_clf.fit(X_train_sub, y_train_sub)
y_pred = l_clf.predict(X_train_sub)
conf_m = confusion_matrix(y_train_sub, y_pred)
#Accuracy
print("Training accuracy of MLPClassifier : ", accuracy(conf_m))
```

```
Training accuracy of MLPClassifier :  0.6873065015479877
```

From above we can see Glucose has strong prediction power for outcome. 75.2% vs. 68.7%. Training accuracy reduced almost 9%. Same was seen with correlation map before model fit.

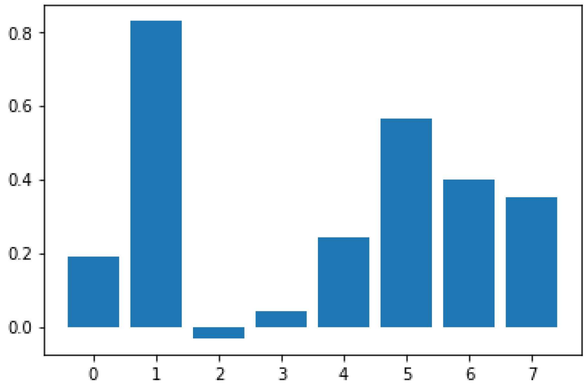
In [ ]:

```
#Logistic regression model feature importance coefficiency for all the features.
importance = l_clf.coef_
results = [item for elem in importance for item in elem]
results
```

In [199]:

```
# Plot feature importance. This is for all the features. Total 8. Top three are Glucose, BMI
#DiabetesPedigreeFunction.
for i,v in enumerate(results):
    print('Feature: %0d, Score: %.5f, Name: %s' % (i,v,features.columns[i]))
# plot feature importance
plt.bar([x for x in range(len(results))], results)
plt.show()
```

```
Feature: 0, Score: 0.19070, Name: Pregnancies
Feature: 1, Score: 0.82807, Name: Glucose
Feature: 2, Score: -0.03223, Name: BloodPressure
Feature: 3, Score: 0.04345, Name: SkinThickness
Feature: 4, Score: 0.24370, Name: Insulin
Feature: 5, Score: 0.56441, Name: BMI
Feature: 6, Score: 0.39936, Name: DiabetesPedigreeFunction
Feature: 7, Score: 0.34920, Name: Age
```



Top three are 1, 5 and 6. Glucose, BMI and DiabetesPedigreeFunction.

Let's use lowest three only. Pregnancies, bloodpressure and skinthickness.

In [200]:

```
#Drop features with low coefficiency. Fit the model.
features_sel = clean_data.drop(columns=['Glucose',
                                'Insulin'
                                'BMI',
                                'DiabetesPedigreeFunction',
                                'Age'
                                'Outcome'])
```

In [202]:

```
X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(features_sel,
                                                                    labels,
                                                                    test_size=0.4,
                                                                    random_state=24)
```

In [203]:

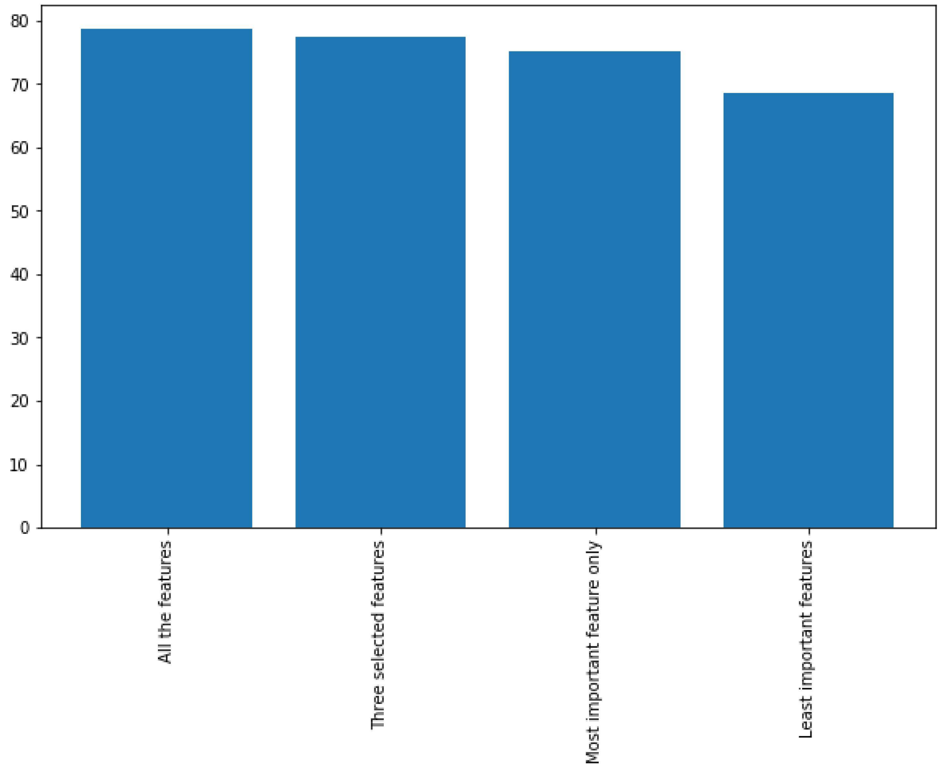
```
#Fit the model
l_clf.fit(X_train_sub, y_train_sub)
y_pred = l_clf.predict(X_train_sub)
conf_m = confusion_matrix(y_train_sub, y_pred)
#Accuracy
print("Training accuracy of MLPClassifier : ", accuracy(conf_m))
```

```
Training accuracy of MLPClassifier :  0.6873065015479877
```

Accuracy went down from 78.6% to 68.7% when most important features are excluded. Accuracy reduced 12.6%.

In [223]:

```
#Plot accuracy for all the cases tested.
results = {'All the features': 78.6,'Three selected features': 77.4,
          'Most important feature only': 75.2, 'Least important features': 68.7}
#plt.figure(figsize=(5.5,3),dpi=120)
plt.figure(figsize=(10,6))
plt.bar(range(len(results)), list(results.values()), align='center')
plt.xticks(range(len(results)), list(results.keys()),rotation ='vertical')
plt.show()
```



NN and permutation

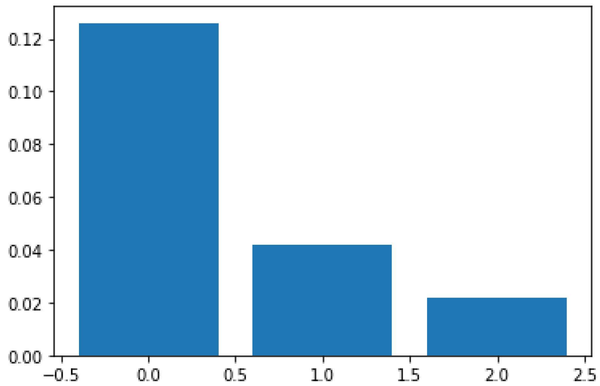
In [91]:

```
#Neural network feature importance
from sklearn.inspection import permutation_importance
NN_result = permutation_importance(clf_sub, X_train_sub, y_train_sub, n_repeats=10, random_st
NN_mean = NN_result.importances_mean
NN_std = NN_result.importances_std
```

In [92]:

```python
#Plot NN feature importance from permutation. Mean.
for i,v in enumerate(NN_mean):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
plt.bar([x for x in range(len(NN_mean))], NN_mean)
plt.show()
```

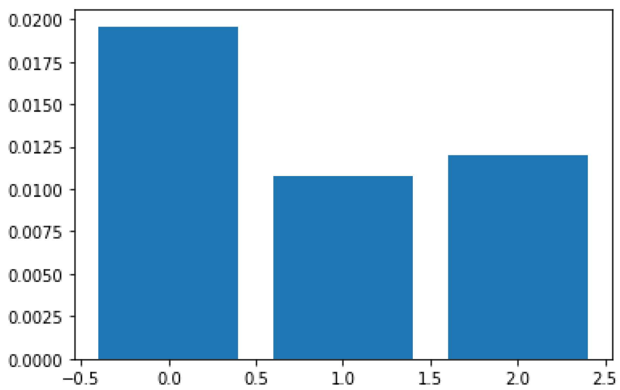
```
Feature: 0, Score: 0.12601
Feature: 1, Score: 0.04211
Feature: 2, Score: 0.02198
```



In [94]:

```python
#Plot NN feature importance from permutation. STD.
for i,v in enumerate(NN_std):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
plt.bar([x for x in range(len(NN_std))], NN_std)
plt.show()
```

```
Feature: 0, Score: 0.01958
Feature: 1, Score: 0.01074
Feature: 2, Score: 0.01195
```



NN permutation indicates same feature importance as with logical regression model.