Report IDS 594
Machine Learning with Python

Shubham Puri

1) Cleaning data and visualization

- Use pandas.read_excel to read the heart_disease.xlsx file into a dataframe called data

```
data = '/Users/shubpuri/Downloads/heart_disease.xlsx'
df1=pd.read_excel(data)
```

- Look at the first 5 rows by using head(5)

```
df.head()
```

- Get the statistics of your dataframe

```
df.describe()
```

- Are there any columns that contain a missing value? If yes, substitute those with the mean value of each column!

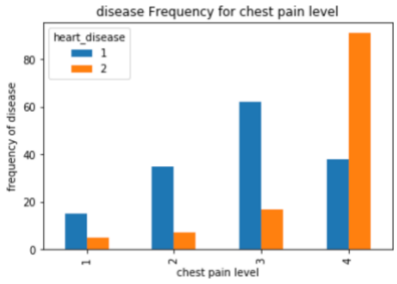
```
df.isna().sum().sort_values(ascending=False)
```

```
heart_disease    0
thal             0
vessel_num       0
slope            0
oldpeak          0
angia            0
heart_rate       0
ecg              0
blood_sugar      0
chol             0
bp               0
chest_pain       0
sex              0
age              0
dtype: int64
```

There are no missing values in the data

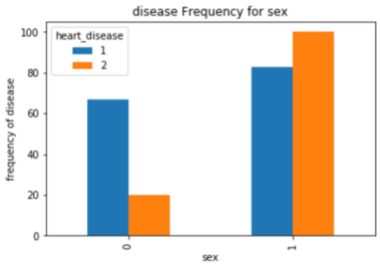
- Look at the disease Frequency for chest pain level (do this for other categorical variables to see if they are useful features)

```
%matplotlib inline
pandas.crosstab(df.chest_pain,df.heart_disease).plot(kind='bar')
plt.title('disease Frequency for chest pain level')
plt.xlabel('chest pain level')
plt.ylabel('frequency of disease')
```



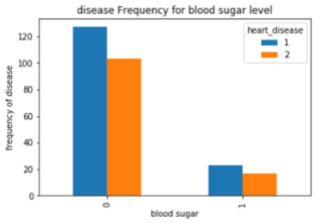
chest pain level 4 has the maximum presence of heart disease While Chest pain level 3 has most people with absence of heart disease

```
%matplotlib inline
pandas.crosstab(df.sex,df.heart_disease).plot(kind='bar')
plt.title('disease Frequency for sex')
plt.xlabel('sex')
plt.ylabel('frequency of disease')
```



maximum heart disease is present for sex 1

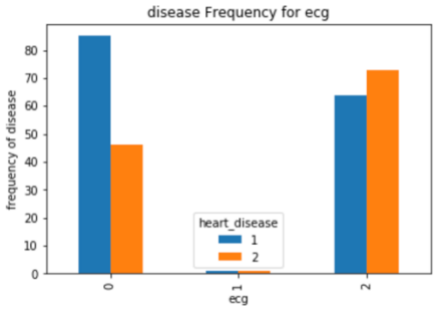
```
%matplotlib inline
pandas.crosstab(df.blood_sugar,df.heart_disease).plot(kind='bar')
plt.title('disease Frequency for blood sugar level')
plt.xlabel('blood sugar')
plt.ylabel('frequency of disease')
```



Most population has blood sugar level 0 out of which more number of people do not have heart disease.

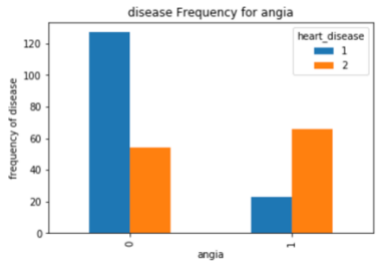
```
%matplotlib inline
pandas.crosstab(df.ecg,df.heart_disease).plot(kind='bar')
plt.title('disease Frequency for ecg')
plt.xlabel('ecg')
```

```
plt.ylabel('frequency of disease')
```



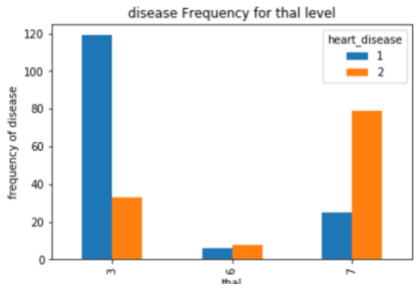
Most population with ECG 0 do not have heart disease while population with ECG 2 have more heart disease. however, Off the less population who has ECG level 1, an equal ratio has heart disease than the ones who don't.

```
%matplotlib inline
pandas.crosstab(df.angia,df.heart_disease).plot(kind='bar')
plt.title('disease Frequency for angia')
plt.xlabel('angia')
plt.ylabel('frequency of disease')
```



The dominating angia 0 have more people with no heart disease while out of the population who have angia 1, there are more cases with a heart disease.

```
%matplotlib inline
pandas.crosstab(df.thal,df.heart_disease).plot(kind='bar')
plt.title('disease Frequency for thal level')
plt.xlabel('thal')
plt.ylabel('frequency of disease')
```



People who have Thal level 3 are less likely to have a heart disease but people who have Thal 7 are very much more likely to have a heart disease.

- Use pandas.get_dummies and put the column names for the categorical variables (refer to the documentation to find which columns are categorical variables). This will create a dummy variable for each level of categories.
- Create a set of dummy variables from the Cat variables

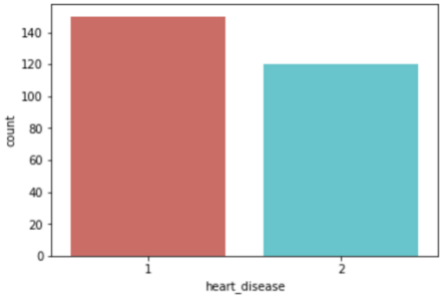
```
df_chestpain = pd.get_dummies(df['chest_pain'])
df_chestpain = df_chestpain.rename(columns = {1:"CP1", 2:"CP2", 3:"CP3", 4:"CP4"})
df_ecg = pd.get_dummies(df['ecg'])
df_ecg = df_ecg.rename(columns = {0:"ECG 0", 1:"ECG 1", 2:"ECG 2"})
df_thal = pd.get_dummies(df['thal'])
df_thal = df_thal.rename(columns = {3:"Thal 3", 6:"Thal 6", 7:"Thal 7"})
df_new = pd.concat([df,df_chestpain,df_ecg, df_thal], axis=1)
#del df_new['chest_pain','blood_sguar','sex','ecg','angia','thal']
df_new = df_new.drop(['chest_pain','ecg','thal'], 1)
```

- What is number of classes disease/not_disease?

```
df_new['heart_disease'].value_counts()
```

1 - 150

2 – 120



- Get your class labels in a variable called y, and remove it from your dataframe . Now, data contains your samples and features, and y contains your labels

```
y = df_new['heart_disease']
df_new.drop('heart_disease', axis=1, inplace=True)
```

- Split the data into a train and validation set. You won't touch the test set until the very end of this program.
  Perform any analysis (model selection, hyperparameter selection) on your training data (X_train_outer)
  DO NOT TOUCH X_test UNTIL THE END OF THIS PROGRAM!!!!

```
from sklearn.model_selection import train_test_split
X_train_outer, X_test, y_train_outer, y_test = train_test_split(df_new, y, test_size=0.2)
print (X_train_outer.shape, y_train_outer.shape)
print (X_test.shape, y_test.shape)
```

```
(216, 20) (216,)
(54, 20) (54,)
```

Classification using nearest neighbor classifier, and see overfitting and underfitting by changing the value of n_neighbors (vary it from 1 to the number of samples -- len(X_train_inner)) What is the accuracy on training and validation when k=1? What about k=len(X_train_inner) ? When underfitting happens when overfitting?

- Split X_train_outer again into X_train_inner, X_val, y_train_inner, y_val (We have an inner training set, and the validation set)

```
X_train_inner, X_val, y_train_inner, y_val = train_test_split(X_train_outer,
y_train_outer, test_size=0.2)
print (X_train_inner.shape, y_train_inner.shape)
print (X_val.shape, y_val.shape)
```

```
(172, 20) (172,)
(44, 20) (44,)
```

- Standardize the training set, and then apply the transformation values to the val set

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train_inner)
X_train_inner_std = scaler.transform(X_train_inner)
X_val_std  = scaler.transform(X_val)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with input dtyp
e uint8, int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: DataConversionWarning: Data with input dtype uint8, i
nt64, float64 were all converted to float64 by StandardScaler.
  after removing the cwd from sys.path.
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: DataConversionWarning: Data with input dtype uint8, i
nt64, float64 were all converted to float64 by StandardScaler.
  """
```

- Fit a K-nearest neighbor classifier model to X_train_inner_std data --> Change the value of n_neighbors here and report the accuracies you get

```
from sklearn.neighbors import KNeighborsClassifier
```

```
classifier = KNeighborsClassifier(n_neighbors=172)
classifier.fit(X_train_inner_std, y_train_inner)
```

output:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',metric_params=None, n_jobs=None, n_neighbors=1, p=2,
           weights='uniform')
```

```
# Print accuracy on your validation data (X_val_std)
classifier.score(X_val_std,y_val)
```

```
0.6818181818181818
```

```
# Print accuracy on your training data (X_train_inner_std)
classifier.score(X_train_inner_std,y_train_inner)
```

```
1.0
```

When n_neighbors=1 , Accuracy on training is 1.0 and validation is 0.68 Which is an example of overfitting

When n_neighbors= 172(Length of X_train_inner), Accuracy on training is 0.58 which is an example of underfitting.

So, how to find best k? Perform 10-fold Cross-validation on your train set to find the best value of k for nearest neighbor classifier (you can also find it by trying different values on the hold-out set you defined above, but since the dataset is small, it is better to perform 10-fold cross-validation, why? because you may by chance get a hold-out validation set that works well with k=1! you never know! So, do it on 10 different val sets and average them to make sure you got a good k!

- In each fold of your cross validation, your training fold needs to be first standardized (scaled), then an algorithm be fit

  Then the same transformation will be applied to each validation fold

  This is done by pipeline and is extremely useful when you use GridSearchCV or cross_val_score and etc..

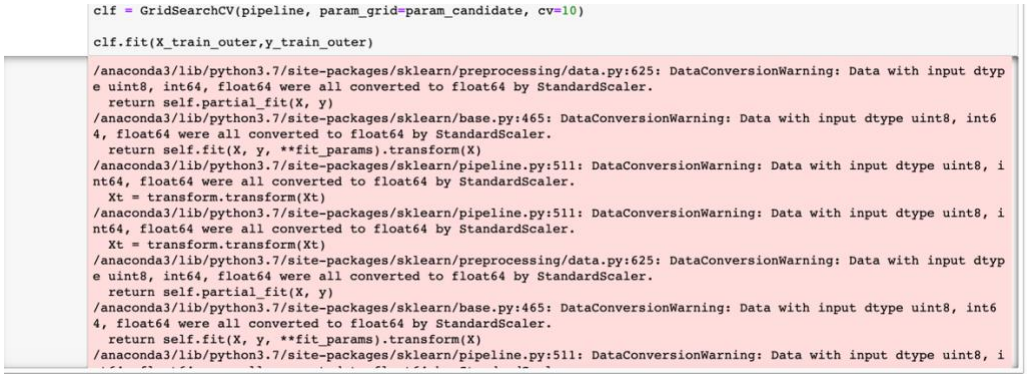
  from sklearn.pipeline import Pipeline

```
scaler = StandardScaler()

clf = KNeighborsClassifier()

pipeline = Pipeline([('transformer', scaler), ('estimator', clf)])
```

- Create a classifier object with GridSearchCV and param_candidate and fit the GridSearchCV object with X_train_outer, and y_train_outer
  Read Python's documentation if anything is unclear!



```
clf = GridSearchCV(pipeline, param_grid=param_candidate, cv=10)

clf.fit(X_train_outer,y_train_outer)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with input dtyp
e uint8, int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/anaconda3/lib/python3.7/site-packages/sklearn/base.py:465: DataConversionWarning: Data with input dtype uint8, int6
4, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, y, **fit_params).transform(X)
/anaconda3/lib/python3.7/site-packages/sklearn/pipeline.py:511: DataConversionWarning: Data with input dtype uint8, i
nt64, float64 were all converted to float64 by StandardScaler.
  Xt = transform.transform(Xt)
/anaconda3/lib/python3.7/site-packages/sklearn/pipeline.py:511: DataConversionWarning: Data with input dtype uint8, i
nt64, float64 were all converted to float64 by StandardScaler.
  Xt = transform.transform(Xt)
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with input dtyp
e uint8, int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/anaconda3/lib/python3.7/site-packages/sklearn/base.py:465: DataConversionWarning: Data with input dtype uint8, int6
4, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, y, **fit_params).transform(X)
/anaconda3/lib/python3.7/site-packages/sklearn/pipeline.py:511: DataConversionWarning: Data with input dtype uint8, i
```

- print clf.best_score_ (refer to Python's documentation for the attributes of clf)

```
print('Best score for data1:', clf.best_score_)
Best score for data1: 0.8379629629629629
```

- View the best parameters for the model found using grid search

```
print('Best k:',clf.best_params_)
Best k: {'estimator__n_neighbors': 57}
```

Perform logistic regression with the default value of C=1, and obtain the cross-validation performance

- Do standardization before classification using Pipeline as Problem 3. You don't need any param_candidate in this case.

```
steps = [('scaler', StandardScaler()), ('logclf', LogisticRegression(C=1))]
pipeline = Pipeline(steps)
```

- Use the cross_val_score on logistic regression on X_train_outer and y_train_outer with cv=10 and get scores

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(pipeline,X_train_outer,y_train_outer,cv=10)
print(scores)
```

```
[0.81818182 0.72727273 0.90909091 0.90909091 0.81818182 0.81818182
 0.9047619  0.76190476 0.9047619  0.85714286]
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with input dtyp
e uint8, int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/anaconda3/lib/python3.7/site-packages/sklearn/base.py:465: DataConversionWarning: Data with input dtype uint8, int6
4, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, y, **fit_params).transform(X)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be ch
anged to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/pipeline.py:511: DataConversionWarning: Data with input dtype uint8, i
nt64, float64 were all converted to float64 by StandardScaler.
  Xt = transform.transform(Xt)
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with input dtyp
e uint8, int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/anaconda3/lib/python3.7/site-packages/sklearn/base.py:465: DataConversionWarning: Data with input dtype uint8, int6
4, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, y, **fit_params).transform(X)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be ch
```

- scores.mean()
  `0.8286363636363637`

Cross-validation performance of which algorithm was better? K-nearest neighbor or logistic regression? You will pick the one that gave you the highest performance (accuracy), and train your model using all training data, test it on your test set (which you never touched up until this point), get your performance, and report it (to me, to the doctor, or the hospital as the generalization performance of your model)

- Standardize the training set (X_train_outer), and then apply the transformation values to the test set (X_test)

```
scaler = StandardScaler().fit(X_train_outer)
X_train_outer_std = scaler.transform(X_train_outer)
X_test_std = scaler.transform(X_test)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with input dtyp
e uint8, int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: DataConversionWarning: Data with input dtype uint8, i
nt64, float64 were all converted to float64 by StandardScaler.
  after removing the cwd from sys.path.
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: DataConversionWarning: Data with input dtype uint8, i
nt64, float64 were all converted to float64 by StandardScaler.
  """
```

```
In [180]: from sklearn.pipeline import Pipeline

          scaler = StandardScaler()
          clf = KNeighborsClassifier(n_neighbors=57)
          pipeline = Pipeline([('transformer', scaler), ('estimator', clf)])
```

```
In [177]: clf.fit(X_train_outer_std,y_train_outer)

Out[177]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                               metric_params=None, n_jobs=None, n_neighbors=57, p=2,
                               weights='uniform')
```

```
In [178]: clf.score(X_test_std,y_test)

Out[178]: 0.9074074074074074
```

```
In [179]: clf.predict(X_test_std)

Out[179]: array([1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 1, 2,
                 1, 1, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1,
                 2, 2, 1, 2, 2, 1, 2, 2, 1, 1])
```

```
In [ ]:
```

KNN performed better than Logistic Regression so we performed KNN on the test data and it gave a score of 0.907.