Algorithmically Predicting Diseases Based on Health History

Shauna Allen, Ben Anderson,
Paul Moses, Sophie Geister-Jones

Initial Postulation: Is it possible to create an algorithm based on pre-existing datasets that then will take individual patient surveys and predict with at least 75% accuracy if that patient is at risk for developing that specific condition?

Questions We Asked

What works best for what we want?

- Neural Network
- Random Forest
- K Nearest Neighbors
- Pipelines (and One Hot Encoder)

- Does using disease-specific data improve the predictiveness of a model vs a more general model?
- Do different modeling techniques work better on different datasets, or does the same method tend to work best on each one?
- Based on these models, which diseases or conditions are more/less easy to predict based on patient histories?
- Does a dataset that's limited to a specific population have more predictive power than one tuned to the general public?

Our First Attempt: Neural Networks

In [32]:
```
!pip install -q -U keras-tuner
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/129.1 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.7/129.1 kB 2.0 MB/s eta 0:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.1/129.1 kB 2.5 MB/s eta 0:00:00
```

In [33]:
```
def create_model(hp):
    nn = tf.keras.models.Sequential()
    # Allow kerastuner to decide which activation function to use in hidden layers
    activation = hp.Choice('activation',['relu','tanh','sigmoid'])
    # Allow kerastuner to decide number of neurons in first layer
    nn.add(tf.keras.layers.Dense(units=hp.Int('first_units',
        min_value=1,
        max_value=26,
        step=2), activation=activation, input_dim=len(X_train[0])))
    # Allow kerastuner to decide number of hidden layers and neurons in hidden layers
    for i in range(hp.Int('num_layers', 1, 6)):
        nn.add(tf.keras.layers.Dense(units=hp.Int('units_' + str(i),
            min_value=1,
            max_value=26,
            step=2),
            activation=activation))
    nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))
    # Compile the model
    nn.compile(loss="binary_crossentropy", optimizer='adam', metrics=["accuracy"])
    return nn

import keras_tuner as kt
tuner = kt.Hyperband(
    create_model,
    objective="val_accuracy",
    max_epochs=20,
    hyperband_iterations=2,
    directory='./untitled_project',
    project_name='tuner1.json')
```

In [34]:
```
tuner.search(X_train_scaled,y_train,epochs=20,validation_data=(X_test_scaled,y_test))
```

```
Trial 60 Complete [00h 00m 06s]
val_accuracy: 0.8235294222831726

Best val_accuracy So Far: 0.8382353186607361
Total elapsed time: 00h 03m 16s
```

In [36]:
```
best_hyper = tuner.get_best_hyperparameters(1)[0]
best_hyper.values
```

Out[36]:
```
{'activation': 'tanh',
 'first_units': 7,
 'num_layers': 6,
 'units_0': 23,
 'units_1': 15,
 'units_2': 3,
 'units_3': 11,
 'units_4': 5,
 'units_5': 25,
 'tuner/epochs': 20,
 'tuner/initial_epoch': 7,
 'tuner/bracket': 2,
 'tuner/round': 2,
 'tuner/trial_id': '0013'}
```

In [37]:
```
best_model = tuner.get_best_models(1)[0]
model_loss, model_accuracy = best_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
3/3 - 0s - loss: 0.4391 - accuracy: 0.8382 - 452ms/epoch - 151ms/step
Loss: 0.43912801146507263, Accuracy: 0.8382353186607361
```

Pivot: Random Forests

```python
conn = sqlite3.connect('Healthcare_Data.sqlite')
query = "SELECT * FROM Heart_Disease"
df_heart = pd.read_sql_query(query, conn)
conn.close
df_heart.head()

X = df_heart.copy()
X.drop(["Heart Disease", "index"], axis=1, inplace=True)
X.head()

dummy_df = pd.get_dummies(df_heart['Heart Disease'])
y = dummy_df['Presence'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=8)
scaler = StandardScaler()
X_scaler = scaler.fit(X_train)
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
rf_model = RandomForestClassifier(n_estimators=500, random_state=78)
rf_model = rf_model.fit(X_train_scaled, y_train)
predictions = rf_model.predict(X_test_scaled)
# Calculating the confusion matrix
cm = confusion_matrix(y_test, predictions)
cm_df = pd.DataFrame(
    cm, index=["Actual 0", "Actual 1"], columns=["Predicted 0", "Predicted 1"]
)

# Calculating the accuracy score
acc_score = accuracy_score(y_test, predictions)

print("Confusion Matrix")
display(cm_df)
print(f"Accuracy Score : {acc_score}")
print("Classification Report")
print(classification_report(y_test, predictions))
```

✓ 0.7s

```
Confusion Matrix
```

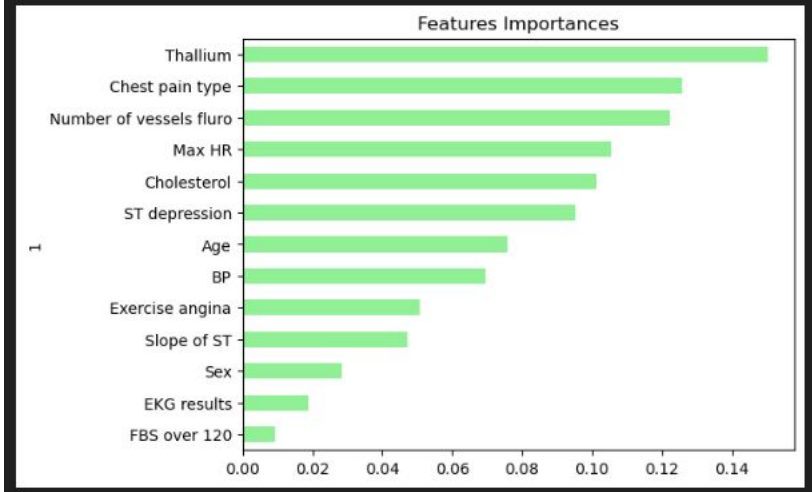
	Predicted 0	Predicted 1
Actual 0	34	6
Actual 1	8	20

```
Accuracy Score : 0.7941176470588235
Classification Report
              precision    recall  f1-score   support

       False       0.81      0.85      0.83        40
        True       0.77      0.71      0.74        28

    accuracy                           0.79        68
   macro avg       0.79      0.78      0.79        68
weighted avg       0.79      0.79      0.79        68
```

```python
# Visualize the features by importance
importances_df = pd.DataFrame(sorted(zip(rf_model.feature_importances_, X.columns), reverse=True))
importances_df.set_index(importances_df[1], inplace=True)
importances_df.drop(columns=1, inplace=True)
importances_df.rename(columns={0: 'Feature Importances'}, inplace=True)
importances_sorted = importances_df.sort_values(by='Feature Importances')
importances_sorted.plot(kind='barh', color='lightgreen', title= 'Features Importances', legend=False)
```

✓ 0.2s

```
<Axes: title={'center': 'Features Importances'}, ylabel='1'>
```



Exploring New Things: Pipeline

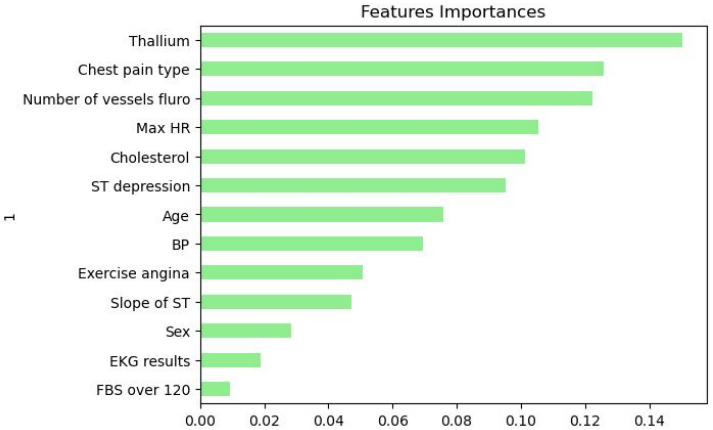
In [6]:
```
numeric_transformer = Pipeline(
    steps=[("imputer", SimpleImputer(strategy="median")), ("scaler", StandardScaler())]
)

categorical_transformer = Pipeline(
    steps=[
        ("encoder", OneHotEncoder(categories='auto')),
        ("selector", SelectPercentile(chi2, percentile=50)),
    ]
)
```

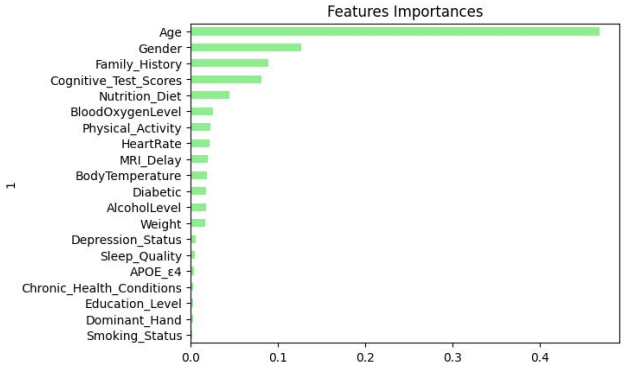
In [7]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_features),
        ("cat", categorical_transformer, cat_transform_features),
    ]
)
```

In [8]:
```
rf = Pipeline(
    steps=[("preprocessor", preprocessor), ("classifier", RandomForestClassifier(n_estimators=30, random_state=78
)

rf_model = rf.fit(X_train, y_train)
print("model score: %.3f" % rf_model.score(X_test, y_test))
```

```
model score: 1.000
```

What features are weighted the most? Is it possible to create an accurate algorithm without them?



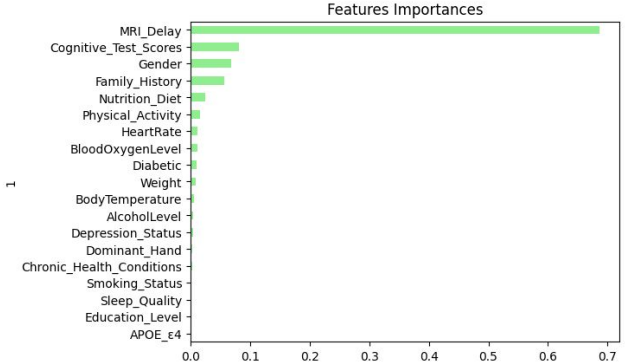


Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	129	0
Actual 1	2	119

Accuracy Score : 0.992
Classification Report

	precision	recall	f1-score	support
0	0.98	1.00	0.99	129
1	1.00	0.98	0.99	121
accuracy			0.99	250
macro avg	0.99	0.99	0.99	250
weighted avg	0.99	0.99	0.99	250



Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	129	0
Actual 1	0	121

Accuracy Score : 1.0
Classification Report

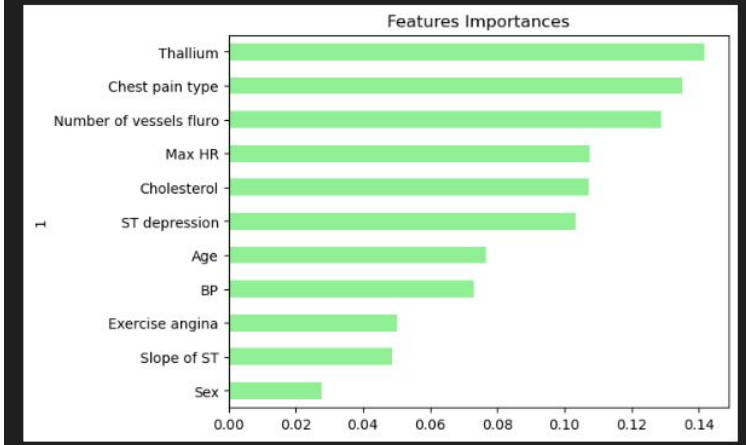
	precision	recall	f1-score	support
0	1.00	1.00	1.00	129
1	1.00	1.00	1.00	121
accuracy			1.00	250
macro avg	1.00	1.00	1.00	250
weighted avg	1.00	1.00	1.00	250

Random Forest 2.0

```python
# Visualize the features by importance
importances_df2 = pd.DataFrame(sorted(zip(rf_model.feature_importances_, X2.columns), reverse=True))
importances_df2.set_index(importances_df2[1], inplace=True)
importances_df2.drop(columns=1, inplace=True)
importances_df2.rename(columns={0: 'Feature Importances'}, inplace=True)
importances_sorted2 = importances_df2.sort_values(by='Feature Importances')
importances_sorted2.plot(kind='barh', color='lightgreen', title= 'Features Importances', legend=False)
```

✓ 0.2s

```
<Axes: title={'center': 'Features Importances'}, ylabel='1'>
```




```python
X2 = df_heart.copy()
X2.drop(["Heart Disease","EKG results", "FBS over 120", 'index'], axis=1, inplace=True)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y, random_state=8)
scaler = StandardScaler()
X_scaler2 = scaler.fit(X_train2)
X_train_scaled2 = X_scaler2.transform(X_train2)
X_test_scaled2 = X_scaler2.transform(X_test2)

rf_model2 = rf_model.fit(X_train_scaled2, y_train2)
predictions2 = rf_model2.predict(X_test_scaled2)
# Calculating the confusion matrix
cm2 = confusion_matrix(y_test2, predictions2)
cm_df2 = pd.DataFrame(
    cm2, index=["Actual 0", "Actual 1"], columns=["Predicted 0", "Predicted 1"]
)

# Calculating the accuracy score
acc_score2 = accuracy_score(y_test2, predictions2)

print("Confusion Matrix")
display(cm_df2)
print(f"Accuracy Score : {acc_score2}")
print("Classification Report")
print(classification_report(y_test2, predictions2))
```

✓ 0.7s

```
Confusion Matrix
```

	Predicted 0	Predicted 1
Actual 0	34	6
Actual 1	7	21

```
Accuracy Score : 0.8088235294117647
Classification Report
              precision    recall  f1-score   support

       False       0.83      0.85      0.84        40
        True       0.78      0.75      0.76        28

    accuracy                           0.81        68
   macro avg       0.80      0.80      0.80        68
weighted avg       0.81      0.81      0.81        68
```

Exploring Options: K Nearest Neighbors

```
Accuracy Score : 0.8088235294117647
Classification Report
              precision    recall  f1-score   support

       False       0.83      0.85      0.84        40
        True       0.78      0.75      0.76        28

    accuracy                           0.81        68
   macro avg       0.80      0.80      0.80        68
weighted avg       0.81      0.81      0.81        68
```

```
from sklearn.neighbors import KNeighborsClassifier
modelKNN = KNeighborsClassifier(n_neighbors=3)
modelKNN.fit(X_train_scaled2, y_train2)
y_pred = modelKNN.predict(X_test_scaled2)

print(classification_report(y_pred,y_test2))
```

```
              precision    recall  f1-score   support

       False       0.85      0.89      0.87        38
        True       0.86      0.80      0.83        30

    accuracy                           0.85        68
   macro avg       0.85      0.85      0.85        68
weighted avg       0.85      0.85      0.85        68
```

Answering our questions

- Does using disease-specific data improve the predictiveness of a model vs a more general model?
- Do different modeling techniques work better on different datasets, or does the same method tend to work best on each one?
- Based on these models, which diseases or conditions are more/less easy to predict based on patient histories?
- Does a dataset that's limited to a specific population have more predictive power than one tuned to the general public?

- We didn't end up exploring more general data
- Neural networks worked really well on the heart disease, but not more so or necessarily waaaaay better than other models that gave us the same/comparable accuracy and potential for more gleaned information re: the model
- Dementia was highly accurate and incredibly easy to predict, no matter how many or how few modifiers were used.
- Based on the RF model run on Pima Indigenous women for diabetes, a more specified population did not seem to have more accuracy then more general numbers. But, the features were not the same so it was not a 1:1 comparison and falls into its own pitfalls.

Pima Women Diabetes Model

```
Accuracy Score : 0.78125
Classification Report
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       129
           1       0.68      0.62      0.65        63

    accuracy                           0.78       192
   macro avg       0.75      0.74      0.75       192
weighted avg       0.78      0.78      0.78       192
```

Initial Postulation: Is it possible to create an algorithm based on pre-existing datasets that then will take individual patient surveys and predict with at least 75% accuracy if that patient is at risk for developing that specific condition?

Where we able to successfully do this?

Yes.