3.) Take the MNIST Fashion data set from earlier in the semester, and:

```
a.) Use hyperparameter based methods to tune a keras based classifier for this data set.

Turn in a pdf of the Jupyter notebook showing this.
```

4.) Build three classifiers (logistic, KNN, neural net) for the Wisconsin breast cancer data set (https://scikit-learn.org/stable/datasets/toy_dataset.html)

And then use an ensemble method to combine the three predictors. Determine the accuracy and AUC for each method alone and then for the ensemble. Use SHAP to explain what the ensemble is doing.

Turn in a notebook showing this

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras import layers

 #a.) Use hyperparameter based methods to tune a keras based classifier for this data set.

#           Turn in a pdf of the Jupyter notebook showing this.
```

Get the files in

```
train_infile="fashion-mnist_train.csv"

test_infile="fashion-mnist_test.csv"

train_df=pd.read_csv(train_infile)

test_df=pd.read_csv(test_infile)
```

Already split into training and test set so just seperate them into the X or input and Y or output.

```
y_train=train_df.pop('label')
X_train=train_df
```

```
y_test=test_df.pop('label')
X_test=test_df
```

```
y_train.shape
```

```
(60000,)
```

```
X_train.shape
```

```
(60000, 784)
```

```
%matplotlib inline
```

```
y_test.head()
```

```
0    0
1    1
2    2
3    2
4    3
Name: label, dtype: int64
```

```
#We have  to one hot encode it
```

```
X_train.head()
```

| | pixel1 | pixel2 | pixel3 | pixel4 | pixel5 | pixel6 | pixel7 | pixel8 | pixel9 | pixel10 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | ... |
| **3** | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | ... |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |

5 rows × 784 columns

```
X_test.shape
```

```
(10000, 784)
```

```
#should I stndarize the pixeles? I am going to.. the original one that we did with base sklearn we didn't standarize it.
```

```
X_train.shape
```

```
(60000, 784)
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = np.array(X_train.loc[0,:])
some_digit_image = some_digit.reshape(28, 28)

plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```

Keras Model creation for classification

```
network=models.Sequential()
network.add(layers.Dense(128,activation='relu',input_shape=(28*28,)))
network.add(layers.Dense(8,activation='relu'))
network.add(layers.Dense(10,activation='softmax'))
```

```
network.compile(optimizer='rmsprop',
               loss='categorical_crossentropy',
               metrics=['accuracy','AUC'])
```

No need to reshape the images as they are aleady in the flat form, but we need to make the numbers be from 0 to 1 for the machine learning model.

```
X_train=X_train.astype('float32')/255
X_test=X_test.astype('float32')/255



#One hot encode the y

from tensorflow.keras.utils import to_categorical

y_test=to_categorical(y_test)
y_train=to_categorical(y_train)


history=network.fit(X_train,y_train,epochs=50,batch_size=512, validation_data=(X_test,y_test))

# Lets go for 50 epochs
```

```
118/118 [==============================] - 1s 12ms/step - loss: 0.1369 - accuracy: 0.9498 - auc: 0.9984 - val_loss: 0.4368 - val_accu
Epoch 23/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1362 - accuracy: 0.9503 - auc: 0.9984 - val_loss: 0.4155 - val_accu
Epoch 24/50
118/118 [==============================] - 1s 12ms/step - loss: 0.1351 - accuracy: 0.9499 - auc: 0.9985 - val_loss: 0.5885 - val_accu
Epoch 25/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1352 - accuracy: 0.9507 - auc: 0.9984 - val_loss: 0.4850 - val_accu
Epoch 26/50
118/118 [==============================] - 1s 12ms/step - loss: 0.1307 - accuracy: 0.9522 - auc: 0.9986 - val_loss: 0.3973 - val_accu
Epoch 27/50
118/118 [==============================] - 2s 14ms/step - loss: 0.1315 - accuracy: 0.9527 - auc: 0.9985 - val_loss: 0.4361 - val_accu
Epoch 28/50
118/118 [==============================] - 2s 19ms/step - loss: 0.1288 - accuracy: 0.9526 - auc: 0.9986 - val_loss: 0.3788 - val_accu
Epoch 29/50
118/118 [==============================] - 2s 18ms/step - loss: 0.1288 - accuracy: 0.9527 - auc: 0.9986 - val_loss: 0.3865 - val_accu
Epoch 30/50
118/118 [==============================] - 2s 17ms/step - loss: 0.1283 - accuracy: 0.9536 - auc: 0.9985 - val_loss: 0.3781 - val_accu
Epoch 31/50
118/118 [==============================] - 2s 20ms/step - loss: 0.1266 - accuracy: 0.9544 - auc: 0.9987 - val_loss: 0.4837 - val_accu
Epoch 32/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1264 - accuracy: 0.9542 - auc: 0.9986 - val_loss: 0.3959 - val_accu
Epoch 33/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1235 - accuracy: 0.9546 - auc: 0.9987 - val_loss: 0.5243 - val_accu
Epoch 34/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1249 - accuracy: 0.9532 - auc: 0.9987 - val_loss: 0.3937 - val_accu
Epoch 35/50
118/118 [==============================] - 3s 21ms/step - loss: 0.1234 - accuracy: 0.9548 - auc: 0.9987 - val_loss: 0.4122 - val_accu
Epoch 36/50
118/118 [==============================] - 2s 18ms/step - loss: 0.1188 - accuracy: 0.9570 - auc: 0.9988 - val_loss: 0.4637 - val_accu
Epoch 37/50
118/118 [==============================] - 2s 16ms/step - loss: 0.1192 - accuracy: 0.9566 - auc: 0.9989 - val_loss: 0.5380 - val_accu
Epoch 38/50
118/118 [==============================] - 1s 12ms/step - loss: 0.1185 - accuracy: 0.9571 - auc: 0.9987 - val_loss: 0.4244 - val_accu
Epoch 39/50
118/118 [==============================] - 1s 12ms/step - loss: 0.1177 - accuracy: 0.9571 - auc: 0.9987 - val_loss: 0.4148 - val_accu
Epoch 40/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1165 - accuracy: 0.9574 - auc: 0.9988 - val_loss: 0.4528 - val_accu
Epoch 41/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1142 - accuracy: 0.9578 - auc: 0.9988 - val_loss: 0.5654 - val_accu
Epoch 42/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1135 - accuracy: 0.9589 - auc: 0.9988 - val_loss: 0.4326 - val_accu
Epoch 43/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1140 - accuracy: 0.9583 - auc: 0.9988 - val_loss: 0.4450 - val_accu
Epoch 44/50
118/118 [==============================] - 1s 13ms/step - loss: 0.1122 - accuracy: 0.9592 - auc: 0.9989 - val_loss: 0.4364 - val_accu
Epoch 45/50
118/118 [==============================] - 2s 17ms/step - loss: 0.1104 - accuracy: 0.9587 - auc: 0.9989 - val_loss: 0.4098 - val_accu
Epoch 46/50
118/118 [==============================] - 2s 19ms/step - loss: 0.1096 - accuracy: 0.9594 - auc: 0.9989 - val_loss: 0.4356 - val_accu
Epoch 47/50
118/118 [==============================] - 1s 12ms/step - loss: 0.1105 - accuracy: 0.9594 - auc: 0.9989 - val_loss: 0.4510 - val_accu
Epoch 48/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1082 - accuracy: 0.9602 - auc: 0.9989 - val_loss: 0.4487 - val_accu
Epoch 49/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1067 - accuracy: 0.9608 - auc: 0.9989 - val_loss: 0.4459 - val_accu
Epoch 50/50
118/118 [==============================] - 1s 11ms/step - loss: 0.1032 - accuracy: 0.9636 - auc: 0.9990 - val_loss: 0.4497 - val_accu
```

```
params={'batch_size':[512,50, 25],
        'epochs':[30,20,10]
        }
```

```
def build_clf():
  # creating the layers of the NN
  ann = models.Sequential()
  ann.add(layers.Dense(128, activation='relu',input_shape=(28*28,)))
  ann.add(layers.Dense(8, activation='relu'))
  ann.add(layers.Dense(10, activation='softmax'))
  ann.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy','AUC'])
  return ann
```

```
!pip install scikeras

from scikeras.wrappers import KerasClassifier
```

```
Requirement already satisfied: scikeras in /usr/local/lib/python3.10/dist-packages (0.12.0)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.3
```

```
model=KerasClassifier(build_fn=build_clf)
```

```
from sklearn.model_selection import GridSearchCV
```

```
# now fit the dataset to the GridSearchCV object
gs=GridSearchCV(estimator=model, param_grid=params, cv=2)
gs = gs.fit(X_train,y_train,validation_data=(X_test,y_test))
```

```
Epoch 14/20
1200/1200 [==============================] - 7s 5ms/step - loss: 0.2479 - accuracy: 0.9093 - auc: 0.9952 - val_loss: 0.3262 - val_acc
Epoch 15/20
1200/1200 [==============================] - 8s 7ms/step - loss: 0.2411 - accuracy: 0.9128 - auc: 0.9954 - val_loss: 0.3124 - val_acc
Epoch 16/20
1200/1200 [==============================] - 6s 5ms/step - loss: 0.2352 - accuracy: 0.9134 - auc: 0.9956 - val_loss: 0.3464 - val_acc
Epoch 17/20
1200/1200 [==============================] - 8s 7ms/step - loss: 0.2315 - accuracy: 0.9165 - auc: 0.9956 - val_loss: 0.3346 - val_acc
Epoch 18/20
1200/1200 [==============================] - 6s 5ms/step - loss: 0.2260 - accuracy: 0.9175 - auc: 0.9958 - val_loss: 0.3551 - val_acc
Epoch 19/20
1200/1200 [==============================] - 8s 7ms/step - loss: 0.2226 - accuracy: 0.9190 - auc: 0.9959 - val_loss: 0.3169 - val_acc
Epoch 20/20
1200/1200 [==============================] - 6s 5ms/step - loss: 0.2196 - accuracy: 0.9197 - auc: 0.9960 - val_loss: 0.3471 - val_acc
```

```
best_params=gs.best_params_
accuracy=gs.best_score_
```

```
accuracy
```

```
0.8816333333333333
```

```
best_params
```

```
{'batch_size': 50, 'epochs': 20}
```

This is the best, I can perhaps get something even better however it will take hours to get that results, this took 30 mins.

```
dir(gs)
```

```
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_abc_impl',
 '_check_feature_names',
 '_check_n_features',
 '_check_refit_for_multimetric',
 '_estimator_type',
 '_format_results',
 '_get_param_names',
 '_get_tags',
 '_more_tags',
 '_repr_html_',
 '_repr_html_inner',
 '_repr_mimebundle_',
 '_required_parameters',
 '_run_search',
 '_select_best_index',
 '_validate_data',
 '_validate_params',
 'best_estimator_',
 'best_index_',
 'best_params_',
```

```
 'refit_time_',
 'return_train_score',
 'score',
 'score_samples',
 'scorer_',
 'scoring',
 'set_params',
 'transform',
 'verbose']
```