0: Loading the MNIST fashion data set

It is already split out into a train and test set, but the X and y values (label or target) are in single file

In this version, we want to build a multiple category classification system, rather than a binary classifier

The opening section of this notebook is identical to the binary classification assignment

Use the ideas from the in class exercise using the MNIST digits set to build a classifier for the Fashion data with multiple categories

I'm going to get you started here a bit, but pay attention to how I load the data here and the data formats used.

This data came as two csv files, with the filenames as shown

I got the data files from kaggle.com, this data set is widely distributed

I loaded this as a pandas data frame, this is a relatively reliable, easy data frame to use

I think this file has a header

See

https://www.kaggle.com/zalando-research/fashionmnist

load the pandas and numpy libraries

used for the data frame tools (Pandas) and to define matrices and do linear algebra (Numpy)

In [1]:
```
import pandas as pd
import numpy as np
```

The next steps load the test and training data into pandas data frames

Pandas has a dataframe structure much like the R dataframe, or an SQL table

There are many pandas member functions that do useful operations on the data frame, here the read_csv() member function is used to load csv files into data frames.

The infile style variables need to have the full path name to the location of the data files in use

If you are working on a local computer, download thes files and then enter their full file address below, using the same format I used.

If you are using google colab to run this,

```
      -click on the file menu on the
left command bar

      -click on the sample data folder
(or create it)

      -upload the two files to colab's
file storage

             -fashion-mnist_train.csv

             -fashion-mnist_test.csv

       -when the files are uploaded,
right click on them and
          copy the file addresses down
to the infile names below
```

In [2]:

```
# the first two lines are for an upload of the data
#train_infile="D:\\Example_data\\MNIST\\fashion-mni
#test_infile="D:\\Example_data\\MNIST\\fashion-mnis

# the next two lines are in infile names when I ran
train_infile="../Data/fashion-mnist_train.csv"
test_infile="../Data/fashion-mnist_test.csv"
```

```
train_df=pd.read_csv(train_infile)

test_df=pd.read_csv(test_infile)
```

Let's look at the available member function for a pandas data frame

In [3]:
```
dir(test_df)
```

Out[3]:
```
['T',
 '_AXIS_LEN',
 '_AXIS_ORDERS',
 '_AXIS_TO_AXIS_NUMBER',
 '_HANDLED_TYPES',
 '__abs__',
 '__add__',
 '__and__',
 '__annotations__',
 '__array__',
 '__array_priority__',
 '__array_ufunc__',
 '__bool__',
 '__class__',
 '__contains__',
 '__copy__',
 '__dataframe__',
 '__deepcopy__',
 '__delattr__',
 '__delitem__',
 '__dict__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__finalize__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__iand__',
 '__ifloordiv__',
 '__imod__',
 '__imul__',
```

```
 '__init__',
 '__init_subclass__',
 '__invert__',
 '__ior__',
 '__ipow__',
 '__isub__',
 '__iter__',
 '__itruediv__',
 '__ixor__',
 '__le__',
 '__len__',
 '__lt__',
 '__matmul__',
 '__mod__',
 '__module__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
 '__nonzero__',
 '__or__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__rand__',
 '__rdivmod__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rfloordiv__',
 '__rmatmul__',
 '__rmod__',
 '__rmul__',
 '__ror__',
 '__round__',
 '__rpow__',
 '__rsub__',
 '__rtruediv__',
 '__rxor__',
 '__setattr__',
```

```
 '__setitem__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 '__weakref__',
 '__xor__',
 '_accessors',
 '_accum_func',
 '_add_numeric_operations',
 '_agg_examples_doc',
 '_agg_summary_and_see_also_doc',
 '_align_frame',
 '_align_series',
 '_append',
 '_arith_method',
 '_as_manager',
 '_attrs',
 '_box_col_values',
 '_can_fast_transpose',
 '_check_inplace_and_allows_duplicate_labels',
 '_check_inplace_setting',
 '_check_is_chained_assignment_possible',
 '_check_label_or_level_ambiguity',
 '_check_setitem_copy',
 '_clear_item_cache',
 '_clip_with_one_bound',
 '_clip_with_scalar',
 '_cmp_method',
 '_combine_frame',
 '_consolidate',
 '_consolidate_inplace',
 '_construct_axes_dict',
 '_construct_result',
 '_constructor',
 '_constructor_sliced',
 '_create_data_for_split_and_tight_to_dict',
 '_data',
```

```
                 '_dir_additions',
                 '_dir_deletions',
                 '_dispatch_frame_op',
                 '_drop_axis',
                 '_drop_labels_or_levels',
                 '_ensure_valid_index',
                 '_find_valid_index',
                 '_flags',
                 '_from_arrays',
                 '_get_agg_axis',
                 '_get_axis',
                 '_get_axis_name',
                 '_get_axis_number',
                 '_get_axis_resolvers',
                 '_get_block_manager_axis',
                 '_get_bool_data',
                 '_get_cleaned_column_resolvers',
                 '_get_column_array',
                 '_get_index_resolvers',
                 '_get_item_cache',
                 '_get_label_or_level_values',
                 '_get_numeric_data',
                 '_get_value',
                 '_getitem_bool_array',
                 '_getitem_multilevel',
                 '_getitem_nocopy',
                 '_gotitem',
                 '_hidden_attrs',
                 '_indexed_same',
                 '_info_axis',
                 '_info_axis_name',
                 '_info_axis_number',
                 '_info_repr',
                 '_init_mgr',
                 '_inplace_method',
                 '_internal_names',
                 '_internal_names_set',
                 '_is_copy',
                 '_is_homogeneous_type',
                 '_is_label_or_level_reference',
```

```
'_is_label_reference',
'_is_level_reference',
'_is_mixed_type',
'_is_view',
'_iset_item',
'_iset_item_mgr',
'_iset_not_inplace',
'_item_cache',
'_iter_column_arrays',
'_ixs',
'_join_compat',
'_logical_func',
'_logical_method',
'_maybe_cache_changed',
'_maybe_update_cacher',
'_metadata',
'_mgr',
'_min_count_stat_function',
'_needs_reindex_multi',
'_protect_consolidate',
'_reduce',
'_reduce_axis1',
'_reindex_axes',
'_reindex_columns',
'_reindex_index',
'_reindex_multi',
'_reindex_with_indexers',
'_rename',
'_replace_columnwise',
'_repr_data_resource_',
'_repr_fits_horizontal_',
'_repr_fits_vertical_',
'_repr_html_',
'_repr_latex_',
'_reset_cache',
'_reset_cacher',
'_sanitize_column',
'_series',
'_set_axis',
'_set_axis_name',
```

```
'_set_axis_nocheck',
'_set_is_copy',
'_set_item',
'_set_item_frame_value',
'_set_item_mgr',
'_set_value',
'_setitem_array',
'_setitem_frame',
'_setitem_slice',
'_slice',
'_stat_axis',
'_stat_axis_name',
'_stat_axis_number',
'_stat_function',
'_stat_function_ddof',
'_take',
'_take_with_is_copy',
'_to_dict_of_blocks',
'_to_latex_via_styler',
'_typ',
'_update_inplace',
'_validate_dtype',
'_values',
'_where',
'abs',
'add',
'add_prefix',
'add_suffix',
'agg',
'aggregate',
'align',
'all',
'any',
'apply',
'applymap',
'asfreq',
'asof',
'assign',
'astype',
'at',
```

```
'at_time',
'attrs',
'axes',
'backfill',
'between_time',
'bfill',
'bool',
'boxplot',
'clip',
'columns',
'combine',
'combine_first',
'compare',
'convert_dtypes',
'copy',
'corr',
'corrwith',
'count',
'cov',
'cummax',
'cummin',
'cumprod',
'cumsum',
'describe',
'diff',
'div',
'divide',
'dot',
'drop',
'drop_duplicates',
'droplevel',
'dropna',
'dtypes',
'duplicated',
'empty',
'eq',
'equals',
'eval',
'ewm',
'expanding',
```

```
 'explode',
 'ffill',
 'fillna',
 'filter',
 'first',
 'first_valid_index',
 'flags',
 'floordiv',
 'from_dict',
 'from_records',
 'ge',
 'get',
 'groupby',
 'gt',
 'head',
 'hist',
 'iat',
 'idxmax',
 'idxmin',
 'iloc',
 'index',
 'infer_objects',
 'info',
 'insert',
 'interpolate',
 'isetitem',
 'isin',
 'isna',
 'isnull',
 'items',
 'iterrows',
 'itertuples',
 'join',
 'keys',
 'kurt',
 'kurtosis',
 'label',
 'last',
 'last_valid_index',
 'le',
```

```
                 'loc',
                 'lt',
                 'mask',
                 'max',
                 'mean',
                 'median',
                 'melt',
                 'memory_usage',
                 'merge',
                 'min',
                 'mod',
                 'mode',
                 'mul',
                 'multiply',
                 'ndim',
                 'ne',
                 'nlargest',
                 'notna',
                 'notnull',
                 'nsmallest',
                 'nunique',
                 'pad',
                 'pct_change',
                 'pipe',
                 'pivot',
                 'pivot_table',
                 'pixel1',
                 'pixel10',
                 'pixel11',
                 'pixel12',
                 'pixel13',
                 'pixel14',
                 'pixel15',
                 'pixel16',
                 'pixel17',
                 'pixel18',
                 'pixel19',
                 'pixel2',
                 'pixel20',
                 'pixel21',
```

```
                  'pixel22',
                  'pixel23',
                  'pixel24',
                  'pixel25',
                  'pixel26',
                  'pixel27',
                  'pixel28',
                  'pixel29',
                  'pixel3',
                  'pixel30',
                  'pixel31',
                  'pixel32',
                  'pixel33',
                  'pixel34',
                  'pixel35',
                  'pixel36',
                  'pixel37',
                  'pixel38',
                  'pixel39',
                  'pixel4',
                  'pixel40',
                  'pixel41',
                  'pixel42',
                  'pixel43',
                  'pixel44',
                  'pixel45',
                  'pixel46',
                  'pixel47',
                  'pixel48',
                  'pixel49',
                  'pixel5',
                  'pixel50',
                  'pixel51',
                  'pixel52',
                  'pixel53',
                  'pixel54',
                  'pixel55',
                  'pixel56',
                  'pixel57',
                  'pixel58',
```

```
'pixel59',
'pixel6',
'pixel60',
'pixel61',
'pixel62',
'pixel63',
'pixel64',
'pixel65',
'pixel66',
'pixel67',
'pixel68',
'pixel69',
'pixel7',
'pixel70',
'pixel71',
'pixel72',
'pixel73',
'pixel74',
'pixel75',
'pixel76',
'pixel77',
'pixel78',
'pixel79',
'pixel8',
'pixel80',
'pixel81',
'pixel82',
'pixel83',
'pixel84',
'pixel85',
'pixel86',
'pixel87',
'pixel88',
'pixel89',
'pixel9',
'pixel90',
'pixel91',
'pixel92',
'pixel93',
'pixel94',
```

```
 'pixel95',
 'pixel96',
 'pixel97',
 'pixel98',
 'pixel99',
 'plot',
 'pop',
 'pow',
 'prod',
 'product',
 'quantile',
 'query',
 'radd',
 'rank',
 'rdiv',
 'reindex',
 'reindex_like',
 'rename',
 'rename_axis',
 'reorder_levels',
 'replace',
 'resample',
 'reset_index',
 'rfloordiv',
 'rmod',
 'rmul',
 'rolling',
 'round',
 'rpow',
 'rsub',
 'rtruediv',
 'sample',
 'select_dtypes',
 'sem',
 'set_axis',
 'set_flags',
 'set_index',
 'shape',
 'shift',
 'size',
```

```
 'skew',
 'sort_index',
 'sort_values',
 'squeeze',
 'stack',
 'std',
 'style',
 'sub',
 'subtract',
 'sum',
 'swapaxes',
 'swaplevel',
 'tail',
 'take',
 'to_clipboard',
 'to_csv',
 'to_dict',
 'to_excel',
 'to_feather',
 'to_gbq',
 'to_hdf',
 'to_html',
 'to_json',
 'to_latex',
 'to_markdown',
 'to_numpy',
 'to_orc',
 'to_parquet',
 'to_period',
 'to_pickle',
 'to_records',
 'to_sql',
 'to_stata',
 'to_string',
 'to_timestamp',
 'to_xarray',
 'to_xml',
 'transform',
 'transpose',
 'truediv',
```

```
 'truncate',
 'tz_convert',
 'tz_localize',
 'unstack',
 'update',
 'value_counts',
 'values',
 'var',
 'where',
 'xs']
```

In [4]:
```
test_df.columns[0:5]
```

Out[4]:
```
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixe
l4'], dtype='object')
```

In [5]:
```
test_df.shape
```

Out[5]:
```
(10000, 785)
```

In [6]:
```
train_df.shape
```

Out[6]:
```
(60000, 785)
```

Okay, I'm expecting 28 x 28 greyscale images again, we have the first column as the label, the rest of this is the pixels

Most skearn models will accept pandas dataframes as input data, so I don't think we need to do much here except split out the first column as y and the rest of the df as X

pandas has a member function called pop that removes a row from the dataframe. We'll use that to both set y_train equal to the labels, and X_train to the remaining df

In [7]:
```
y_train=train_df.pop('label')
X_train=train_df
```

In [8]:
```
print(y_train.shape)
print(X_train.shape)
```

```
(60000,)
(60000, 784)
```

In [9]:
```
y_test=test_df.pop('label')
X_test=test_df
```

Labels

Each training and test example is assigned to one of the following labels:

0 T-shirt/top 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Sandal 6 Shirt 7 Sneaker 8 Bag 9 Ankle boot

The % symbol indicates that this is a magic function, that is to say a function command for the jupyter notebook server, not to the python kernel

This particular command causes plots created using the matplotlib libary to print in the notebook not in a new window

In [10]:
```
%matplotlib inline
```

1: Data plots

Okay here is the visualization of one image, a shirt

Note: I use a location slice of the X_train dataframe, X_train.loc[0,:] to get row zero, all entries, or the first image in the array. I then force that into the np.array form so I can use the reshape() member function to reshape the row of data into a 28 x 28 image.

```
I don't think that pandas easily
allows the reshape maneuver, so
that's why I converted to an
np.array,  the
reshape operation produces an np
matrix that can be plotted with
imshow.  There may be a better way to
do this.  Hmm.
```

Also I checked, and we can feed X_train into the training input of the classifier as a pd.dataframe, there is no need to change the format

Most sklearn models will accept either pandas dataframes or np matrices as inputs, which is a help

In [11]:
```
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()
```

In [12]:
```
y_train[0]
```

Out[12]: 2

Question/Action

What type of cloting is this image supposed to be?

Insert a cell with your answer below

Question/Action

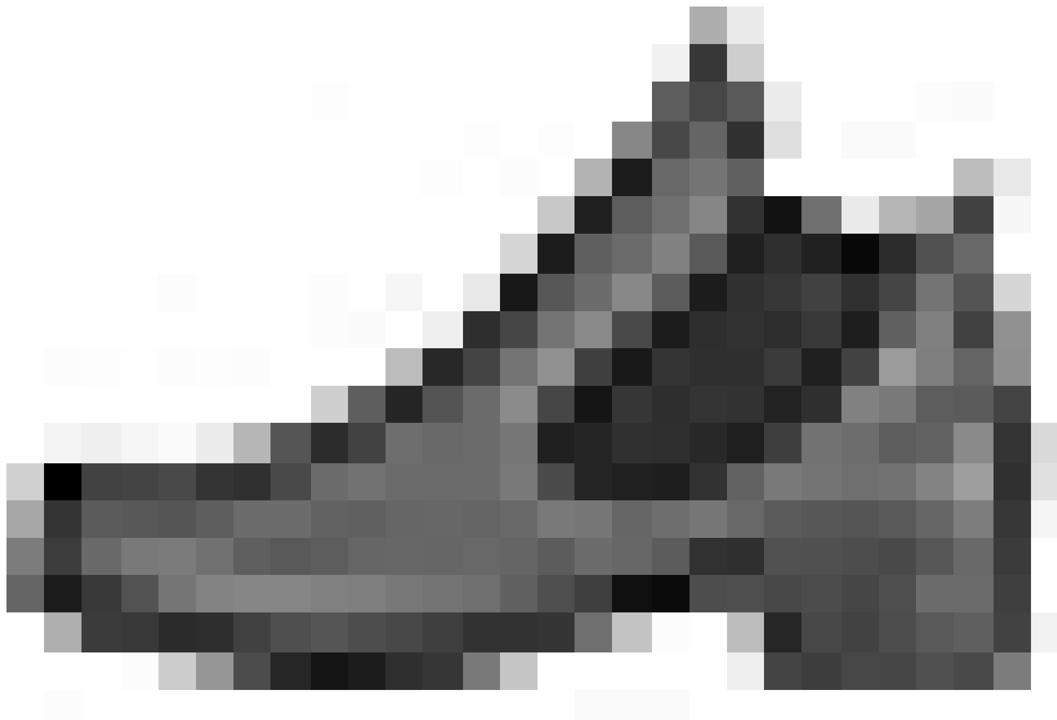
Show images of a sandal and a sneaker from this data set, show them in cells below

Show all your steps

In [13]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = np.array(X_train.loc[1,:])
some_digit_image = some_digit.reshape(28, 28)

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



In [14]:
```
y_train[1]
```

Out[14]: 9

2: Time to Build some models

Lets build a multi category classifying neural net model

fclf - this should classify each image to one of the ten label classes (0-9)

Okay, go build some models- Assignment

1.) For each model find the accuracy, the confusion matrix, the precision and the recall, label these all/ Look at the confusion matrix, explain which classes of objects were most likely to be confused with each other and which were most distinct. Explain why you think this happens, does it make sense?

```
I ran these quickly (so I know this
works) and got 87.7 % accuracy for
the X-train data set using all 10
classes and
97.7 % accuracy  for the binary
classification (ie the pullover
detector).  See if you can beat the
quick results I got.      Post your
results in the discussion section of
D2L for this week.  Discuss what you
did to beat my score
```

2.) Also, create the ROC curve for the binary classifier and compute the AUC for the ROC, for the binary classifier, but not for the 10 element classifier

3.) When you are done with steps 1 and 2, use your two classifier models to classify the test data. Is there evidence of overfitting? What tells you this?

Print your completed jupyter notebook to a pdf file, you can use the browser to print to pdf. Upload this to dropbox in D2L to submit the homework.

fclf2- A binary classifier as either class 2 or not

Since class two is a pullover, this is a "pullover detector"

In [15]:
```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, rand
# the hidden layers were 20,10,5
clf.fit(X_train, y_train)
```

```
Iteration 1, loss = 2.39701701
Iteration 2, loss = 1.26783595
Iteration 3, loss = 0.91331237
Iteration 4, loss = 0.78310348
Iteration 5, loss = 0.69564741
Iteration 6, loss = 0.63831862
Iteration 7, loss = 0.57820165
Iteration 8, loss = 0.53043468
Iteration 9, loss = 0.49799182
Iteration 10, loss = 0.47684637
Iteration 11, loss = 0.46056097
Iteration 12, loss = 0.44871773
Iteration 13, loss = 0.43886652
Iteration 14, loss = 0.43281108
Iteration 15, loss = 0.42625512
Iteration 16, loss = 0.42238268
Iteration 17, loss = 0.41704239
Iteration 18, loss = 0.41061998
Iteration 19, loss = 0.41468647
Iteration 20, loss = 0.40866045
Iteration 21, loss = 0.40560107
Iteration 22, loss = 0.39711162
Iteration 23, loss = 0.39359247
Iteration 24, loss = 0.39429087
Iteration 25, loss = 0.39189282
Iteration 26, loss = 0.38827783
Iteration 27, loss = 0.38641069
Iteration 28, loss = 0.38699566
Iteration 29, loss = 0.38334247
Iteration 30, loss = 0.38240784
Iteration 31, loss = 0.37543925
Iteration 32, loss = 0.37230378
Iteration 33, loss = 0.37446763
Iteration 34, loss = 0.37263191
Iteration 35, loss = 0.36475039
Iteration 36, loss = 0.36682977
Iteration 37, loss = 0.36315167
Iteration 38, loss = 0.36035363
Iteration 39, loss = 0.36522729
Iteration 40, loss = 0.36035252
```

```
Iteration 41, loss = 0.35660430
Iteration 42, loss = 0.35576797
Iteration 43, loss = 0.35553212
Iteration 44, loss = 0.35219978
Iteration 45, loss = 0.35392071
Iteration 46, loss = 0.34836479
Iteration 47, loss = 0.35352245
Iteration 48, loss = 0.34657533
Iteration 49, loss = 0.34845672
Iteration 50, loss = 0.34663882
Iteration 51, loss = 0.34307613
Iteration 52, loss = 0.34147375
Iteration 53, loss = 0.34322582
Iteration 54, loss = 0.33996163
Iteration 55, loss = 0.33942158
Iteration 56, loss = 0.33980426
Iteration 57, loss = 0.33669304
Iteration 58, loss = 0.33866180
Iteration 59, loss = 0.33814570
Iteration 60, loss = 0.33097066
Iteration 61, loss = 0.33314326
Iteration 62, loss = 0.33548404
Iteration 63, loss = 0.33101922
Iteration 64, loss = 0.32778643
Iteration 65, loss = 0.33064806
Iteration 66, loss = 0.33144293
Iteration 67, loss = 0.32442077
Iteration 68, loss = 0.32725587
Iteration 69, loss = 0.32499830
Iteration 70, loss = 0.32549903
Iteration 71, loss = 0.33029769
Iteration 72, loss = 0.32939584
Iteration 73, loss = 0.32100154
Iteration 74, loss = 0.32644517
Iteration 75, loss = 0.32280949
Iteration 76, loss = 0.32113548
Iteration 77, loss = 0.32502426
Iteration 78, loss = 0.31831366
Iteration 79, loss = 0.32028025
Iteration 80, loss = 0.32518164
```

```
Iteration 81, loss = 0.31984769
Iteration 82, loss = 0.31729616
Iteration 83, loss = 0.31996047
Iteration 84, loss = 0.31735705
Iteration 85, loss = 0.31498409
Iteration 86, loss = 0.31721687
Iteration 87, loss = 0.31940991
Iteration 88, loss = 0.31280738
Iteration 89, loss = 0.31110452
Iteration 90, loss = 0.31520596
Iteration 91, loss = 0.31808982
Iteration 92, loss = 0.31642062
Iteration 93, loss = 0.31561801
Iteration 94, loss = 0.31057491
Iteration 95, loss = 0.30957906
Iteration 96, loss = 0.31271667
Iteration 97, loss = 0.30783567
Iteration 98, loss = 0.31435207
Iteration 99, loss = 0.30832317
Iteration 100, loss = 0.30950177
Iteration 101, loss = 0.31304043
Iteration 102, loss = 0.30699052
Iteration 103, loss = 0.30802130
Iteration 104, loss = 0.31065979
Iteration 105, loss = 0.30826191
Iteration 106, loss = 0.30527069
Iteration 107, loss = 0.31191678
Iteration 108, loss = 0.30574304
Iteration 109, loss = 0.30826866
Iteration 110, loss = 0.30698797
Iteration 111, loss = 0.30357593
Iteration 112, loss = 0.30443350
Iteration 113, loss = 0.31114983
Iteration 114, loss = 0.30372815
Iteration 115, loss = 0.30249066
Iteration 116, loss = 0.30263194
Iteration 117, loss = 0.30823893
Iteration 118, loss = 0.30644104
Iteration 119, loss = 0.30109283
Iteration 120, loss = 0.30214095
```

```
Iteration 121, loss = 0.29773336
Iteration 122, loss = 0.30501248
Iteration 123, loss = 0.30265914
Iteration 124, loss = 0.29946483
Iteration 125, loss = 0.29843924
Iteration 126, loss = 0.29944263
Iteration 127, loss = 0.30114820
Iteration 128, loss = 0.29924070
Iteration 129, loss = 0.29931914
Iteration 130, loss = 0.29883424
Iteration 131, loss = 0.30033114
Iteration 132, loss = 0.30006475
Training loss did not improve more than tol=0.00010
0 for 10 consecutive epochs. Stopping.
```

Out[15]:

```
▾ MLPClassifier
MLPClassifier(alpha=1e-05, hidden_layer_size
s=(20, 15, 15, 10), max_iter=500,
              random_state=1, verbose=True)
```

In [16]:

```
clf.predict_proba(X_test.iloc[:10,:])
```

Out[16]:
```
array([[6.72385918e-001, 5.04030646e-007, 1.5167874
9e-005,
         8.76450799e-006, 8.77883579e-007, 4.0760134
0e-024,
         3.27555320e-001, 5.04984645e-060, 3.3448490
3e-005,
         1.08742223e-017],
        [2.00368068e-018, 9.99999997e-001, 1.0314248
3e-043,
         3.76742609e-012, 3.17456073e-009, 8.7270730
9e-075,
         1.75549873e-028, 8.92745988e-136, 1.0043280
3e-024,
         2.83130753e-053],
        [1.46322415e-002, 1.54464640e-004, 8.7271552
4e-001,
         1.36606068e-002, 1.90523850e-002, 2.6109249
7e-010,
         7.88079560e-002, 9.83435557e-018, 9.7115727
8e-004,
         5.66472759e-006],
        [3.06618459e-001, 6.01482318e-004, 1.6978404
2e-001,
         2.00693210e-002, 5.03760022e-003, 1.3961768
7e-010,
         4.92441525e-001, 4.56697738e-022, 5.4469457
4e-003,
         6.25733581e-007],
        [3.63280069e-003, 1.62981810e-003, 1.2668400
0e-002,
         4.53463784e-001, 5.10413010e-001, 1.6692737
0e-039,
         1.80587851e-002, 1.57567182e-072, 1.3340141
8e-004,
         5.82665319e-020],
        [4.05731073e-001, 1.59739933e-003, 6.0573366
3e-002,
         2.31589981e-002, 3.10818324e-003, 1.5197966
6e-008,
         4.92248685e-001, 1.19365896e-018, 1.3580092
```

```
1e-002,
        2.18784425e-006],
       [1.28166515e-002, 3.99654620e-004, 1.1663554
8e-003,
        1.39511067e-003, 1.01117983e-004, 8.0990692
6e-004,
        2.07409434e-002, 6.57290114e-008, 9.6247420
1e-001,
        9.59930483e-005],
       [3.70801177e-002, 3.00548576e-004, 3.7936225
6e-001,
        2.80817089e-002, 1.26513045e-001, 3.6952879
0e-020,
        4.27472913e-001, 6.81081126e-041, 1.1894099
7e-003,
        1.14536662e-011],
       [6.57609972e-049, 4.99268267e-040, 2.3913432
0e-064,
        2.65848532e-054, 2.48704902e-033, 1.0000000
0e+000,
        3.56273515e-057, 1.31023423e-021, 2.7788086
8e-032,
        7.05962423e-018],
       [9.89674232e-001, 1.73858092e-006, 5.4372121
5e-010,
        5.49568889e-006, 1.57673545e-009, 1.1322420
5e-036,
        1.03093553e-002, 9.59327002e-084, 9.1759146
6e-006,
        3.89583055e-025]])
```

In [17]:

```
clf.predict(X_test.iloc[:10,:])
```

Out[17]:

```
array([0, 1, 2, 6, 4, 6, 8, 6, 5, 0], dtype=int64)
```

In [18]:

```
y_test[0:10]
```

Out[18]:
```
0    0
1    1
2    2
3    2
4    3
5    2
6    8
7    6
8    5
9    0
Name: label, dtype: int64
```

In [21]:
```python
y_pred=clf.predict(X_train)
```

In [22]:
```python
from sklearn.metrics import confusion_matrix

my_cm=confusion_matrix(y_train,y_pred)
my_cm
```

Out[22]:
```
array([[5146,   16,   91,  118,   21,    0,  596,
0,   12,    0],
       [   6, 5875,    9,   65,   29,    1,   14,
0,    1,    0],
       [  27,    0, 4775,   35,  908,    0,  251,
0,    4,    0],
       [ 213,   71,   85, 5110,  366,    1,  149,
0,    4,    1],
       [   3,    6,  355,   79, 5386,    3,  168,
0,    0,    0],
       [   0,    0,    1,    0,    0, 5923,    3,
56,    4,   13],
       [ 879,    6,  676,   74,  806,    0, 3548,
0,   11,    0],
       [   0,    0,    0,    0,    0,   47,    0, 5
835,    3,  115],
       [  14,    2,   32,    8,   30,   12,   75,
28, 5799,    0],
       [   0,    0,    0,    0,    0,   12,    0,
214,    0, 5774]],
      dtype=int64)
```

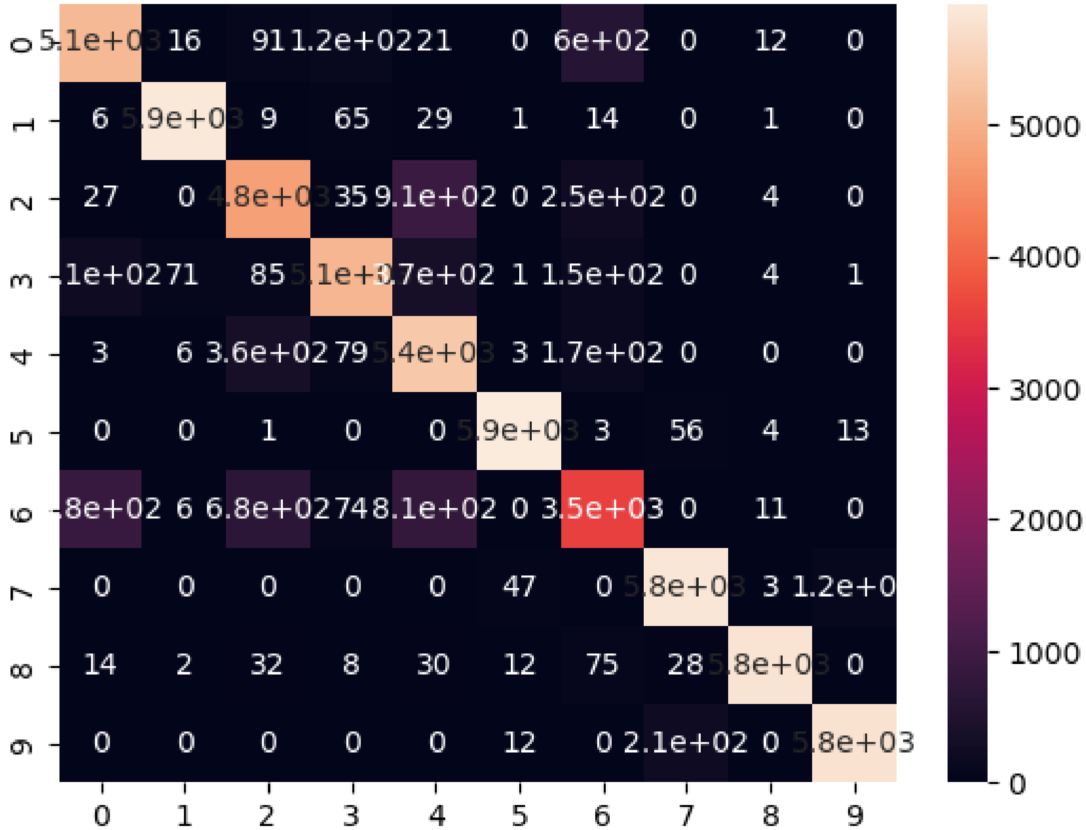
Action

Add the confusion matrix and visualize it as a heatmap

In [24]:
```
import seaborn as sns

sns.heatmap(my_cm,annot=True)
```

Out[24]:
```
<Axes: >
```

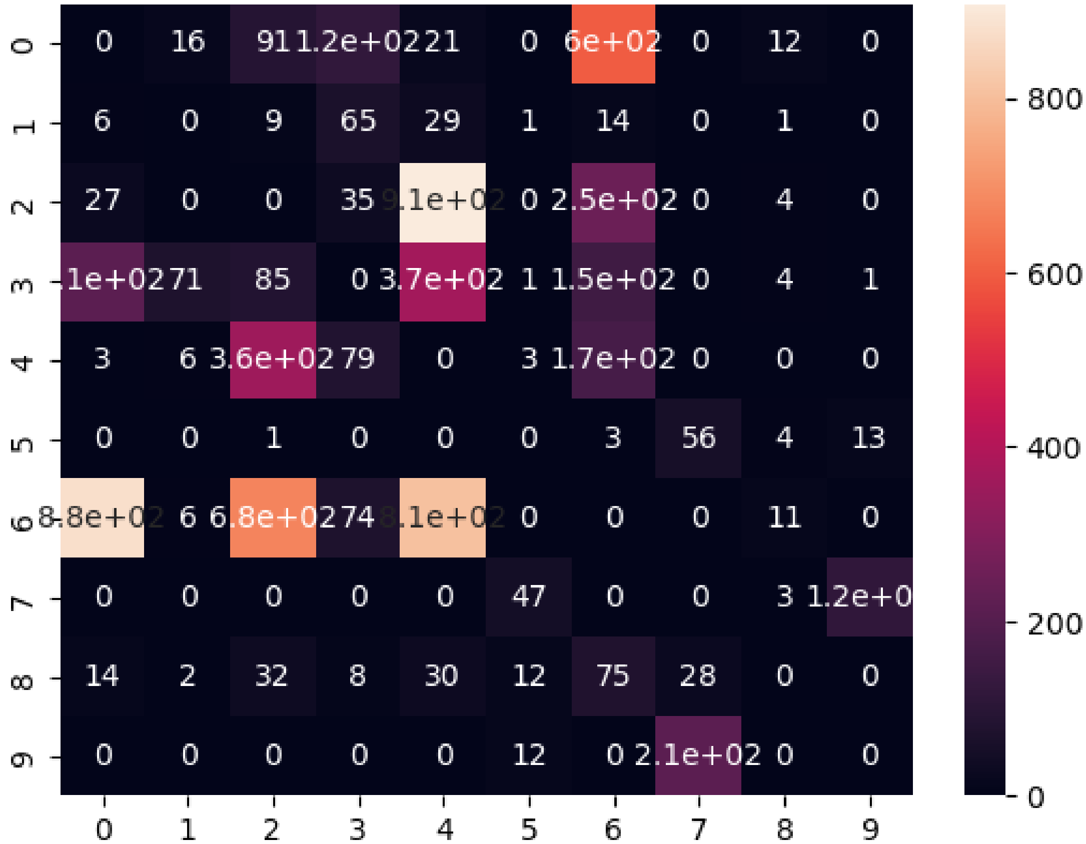


In [ ]:
```
plt.ticklabel_format(useOffset=False)
```

In [25]:
```
import numpy as np

sns.heatmap(my_cm-np.eye(my_cm.shape[0])*my_cm,anno
```

Out[25]: <Axes: >



Action

Figure out how to calculate the percentage of correct answers for each category

In [23]:
```
my_cm.trace()/my_cm.sum()
```

Out[23]:
```
0.8861833333333333
```

In [35]:
```
from sklearn.metrics import classification_report
print(classification_report(y_train, y_pred, digits
```

```
              precision    recall  f1-score   suppo
rt

           0     0.8184    0.8577    0.8376      60
00
           1     0.9831    0.9792    0.9811      60
00
           2     0.7927    0.7958    0.7942      60
00
           3     0.9310    0.8517    0.8895      60
00
           4     0.7138    0.8977    0.7952      60
00
           5     0.9873    0.9872    0.9872      60
00
           6     0.7386    0.5913    0.6568      60
00
           7     0.9514    0.9725    0.9618      60
00
           8     0.9933    0.9665    0.9797      60
00
           9     0.9781    0.9623    0.9702      60
00

    accuracy                         0.8862     600
00
   macro avg     0.8888    0.8862    0.8853     600
00
weighted avg     0.8888    0.8862    0.8853     600
00
```