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

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 us

In [2]:
```
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.
```

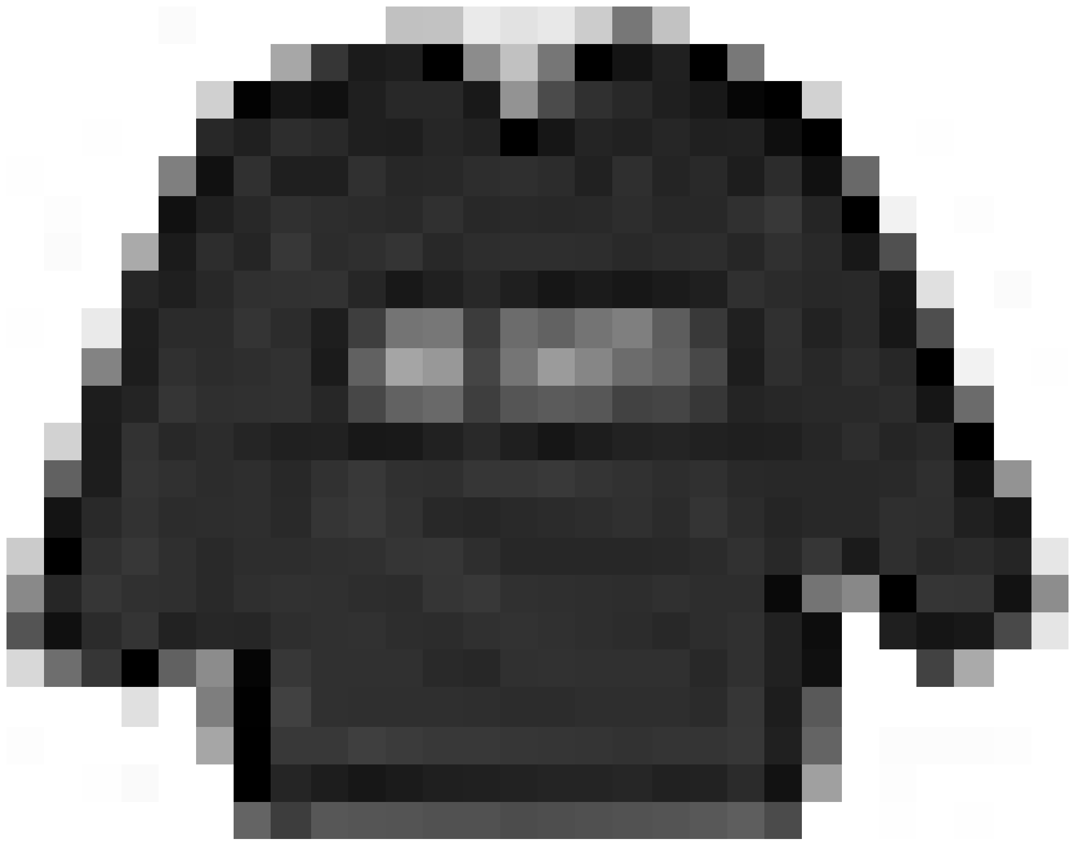
Also I checked, and we can feed X_train into the training input of the classifier as a pd.dataframe, there is no need to change the format

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

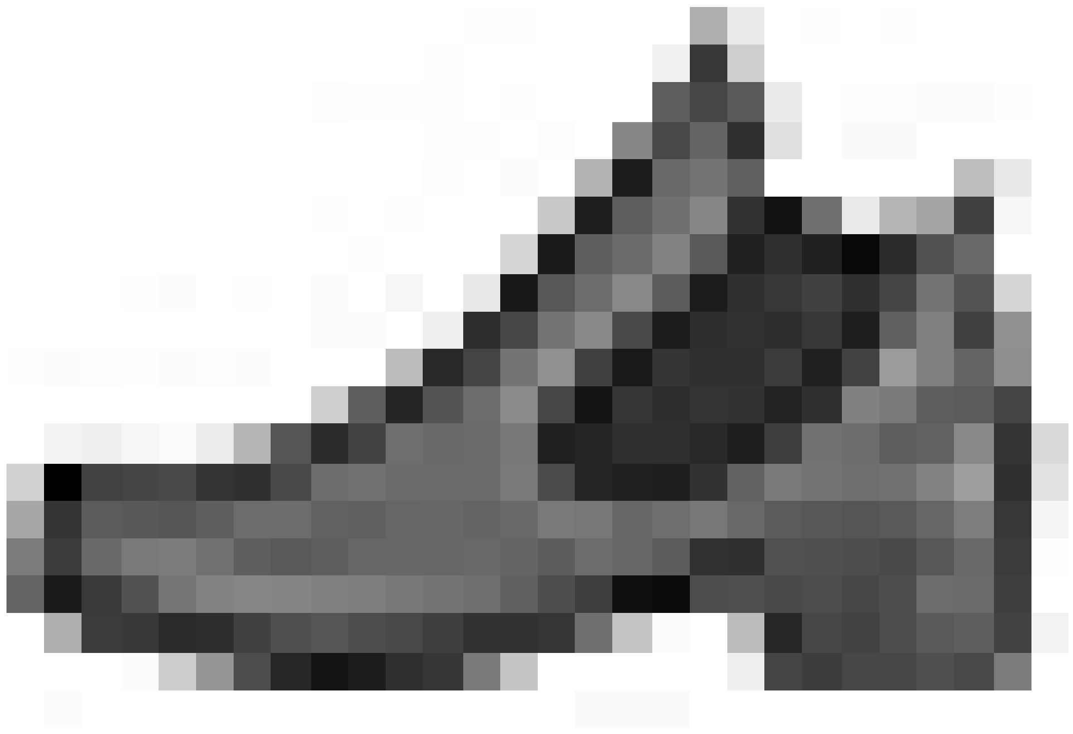
Show images of a sandal and a sneaker from this data set, show them in cells below

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 two different classifying neural net models

fclf - this should classify each image to one of the ten label classes (0-9) fclf2- classify everything as either a pullover (2) or not a pullover, this is a binary categorization

In [15]:
```
y_train_2=(y_train==2)
y_test_2=(y_test==2)
```

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.

Model fclf, classify fashion image to 10 categories

In [16]:
```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, rand
```

In [17]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(clf, X_train, y_train_2, cv=3, scor
# These are the accuracy scores
```

Out[17]:
```
array([0.9566, 0.9555, 0.9578])
```

In [21]:
```
from sklearn.model_selection import cross_val_predi
y_train_pred = cross_val_predict(clf, X_train, y_tr

from sklearn.metrics import precision_score, recall_
print("precision_score: ", precision_score(y_train_
# When predicting postive item the model can correc

print("recall_score: ", recall_score(y_train_2, y_t
# When looking at the actual postive value it can d
```

```
precision_score:  0.7968204053109713
recall_score:  0.7601666666666667
```

Action

show all the performance measures used in the MNIST digit classifier answer seen in class

In [23]:
```
clf.fit(X_train, y_train_2)
pred_test=clf.predict(X_test)

# Accuracy
n_correct = sum(y_test_2 == pred_test)
print(n_correct / len(y_test_2))
```

```
0.9608
```

In [24]:
```
print("precision_score: ", precision_score(y_test_2
# When predicting postive item the model can correc

print("recall_score: ", recall_score(y_test_2, pred
# When looking at the actual postive value it can d
```

```
precision_score:  0.8543123543123543
recall_score:  0.733
```