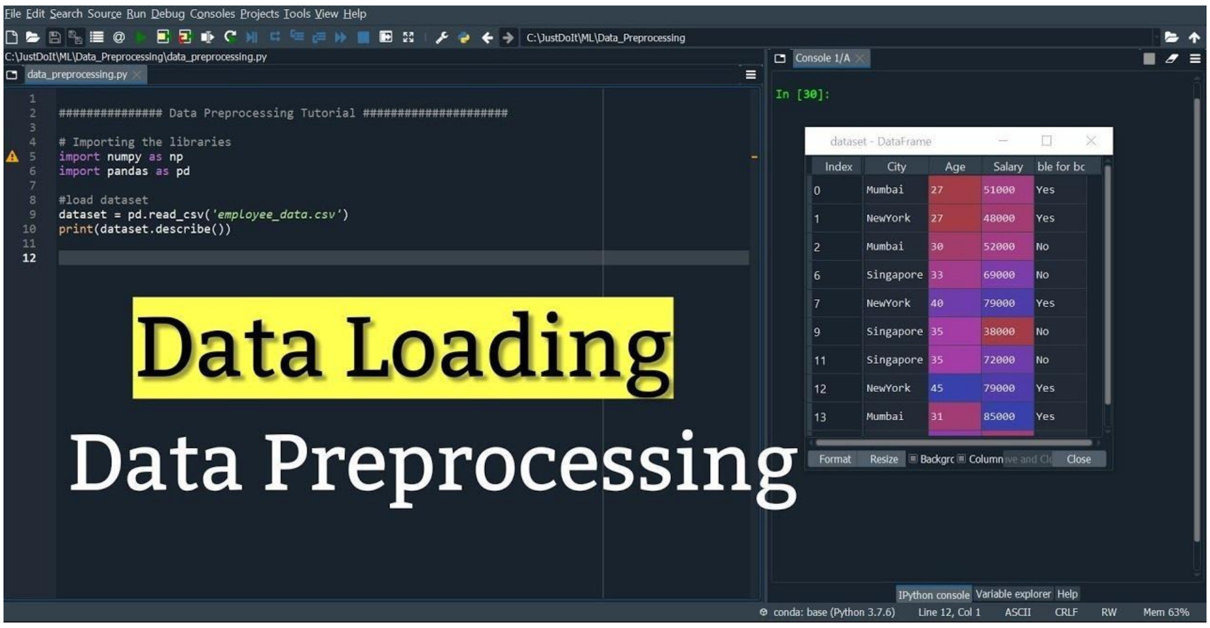
AI-driven exploration and prediction project by loading and preprocessing the dataset.

PHASE 3 DOCUMENTATION

PROJECT: *Dermatology MNIST: Loading and Processing*

Abstract


File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\JustDoIt\ML\Data_Preprocessing
C:\JustDoIt\ML\Data_Preprocessing\data_preprocessing.py
data_preprocessing.py
Data Loading
Data Preprocessing
Console 1/A
dataset - DataFrame
Format
Resize
Backgrc
Column
Close
IPython console
Variable explorer
Help
conda: base (Python 3.7.6)
Line 12, Col 1
ASCII
CRLF
RW
Mem 63%

```
############### Data Preprocessing Tutorial #####################

# Importing the libraries
import numpy as np
import pandas as pd

#load dataset
dataset = pd.read_csv('employee_data.csv')
print(dataset.describe())
```

```
In [30]:
```

Index	City	Age	Salary	ble for bc
0	Mumbai	27	51000	Yes
1	NewYork	27	48000	Yes
2	Mumbai	30	52000	No
6	Singapore	33	69000	No
7	NewYork	40	79000	Yes
9	Singapore	35	38000	No
11	Singapore	35	72000	No
12	NewYork	45	79000	Yes
13	Mumbai	31	85000	Yes

Loading and preprocessing data is a critical step in any machine learning workflow. The goal of loading and preprocessing is to ensure that the data is in a format that is compatible with the machine learning model and that the data is of high quality.

There are a number of different loading and preprocessing methods that can be used, depending on the type of data and the specific machine learning task. However, some common loading and preprocessing steps include:

- Loading the data: The first step is to load the data into a format that can be processed by the machine learning model. This may involve converting the data to a specific data type, such as a NumPy array or a TensorFlowdataset.
- Handling missing values: Missing values are a common problem in real-world data. There are a number of different ways to handle missing values, such as dropping the rows with missing values, imputing the missing values witha

mean or median value, or using a machine learning model to predict the missing values.

- Encoding categorical data: Categorical data, such as text or labels, needs to be encoded into numerical values before it can be used by a machine learning model. There are a number of different ways to encode categorical data, such as one-hot encoding, label encoding, andembedding.
- Scaling the data: Some machine learning models require the data to be scaled to a specific range. This is typically done by subtracting the mean from each feature and dividing by the standard deviation.

Once the data has been loaded and preprocessed, it is ready to be used to train and evaluate a machine learning model.

Here is an example of a simple loading and preprocessing method for a dataset of images:

1. Load the images into NumPy arrays.
2. Resize the images to a consistentsize.
3. Normalize the pixel values to a range of [0,1].

Once the images have been loaded and preprocessed, they can be used to train and evaluate an image classification model.

There are a number of different tools and libraries that can be used to load and preprocess data. Some popular options include:

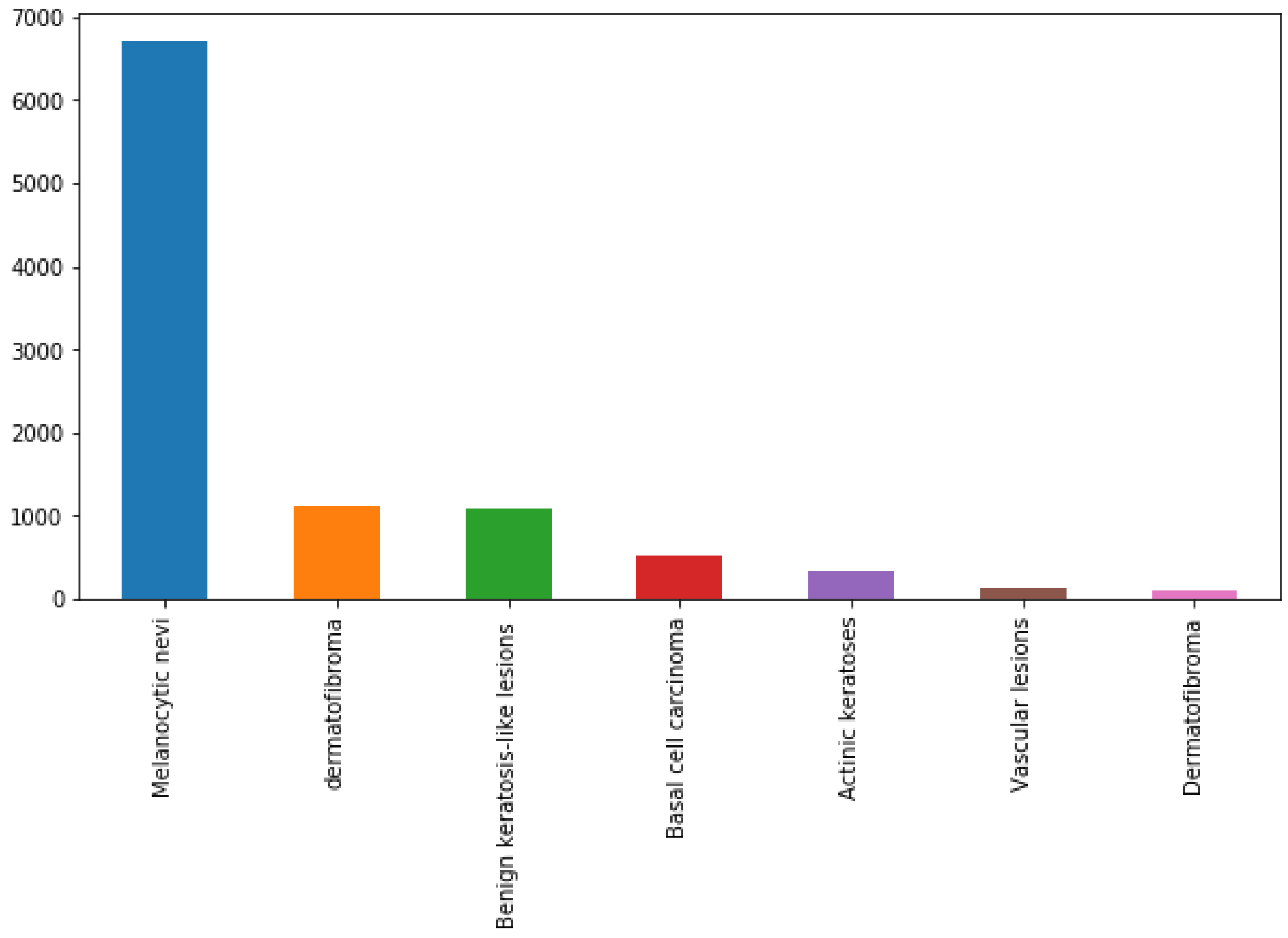
- Python: NumPy, Pandas, TensorFlow, andscikit-learn.
- R: dplyr, tidyr, andcaret.

The specific loading and preprocessing steps that are required will vary depending on the type of data and the specific machine learning task. However, the general principles outlined above are applicable to all machine learning workflows.

PYTHON PROGRAMMING:

```
matplotlib inline
import matplotlib.pyplot as
plt import numpy as np
import pandas as
pd import os
from glob import
glob import seaborn
as sns
base_skin_dir = os.path.join('..', 'input')
imageid_path_dict =
          {os.path.splitext(os.path.basename(x))[0]: x for x in
          glob(os.path.join(base_skin_dir, '*', '*.jpg'))}

lesion_type_dict = {
  'nv': 'Melanocytic nevi',
  'mel': 'dermatofibroma',
  'bkl': 'Benign keratosis-like lesions ',
  'bcc': 'Basal cell carcinoma',
  'akiec': 'Actinic
  keratoses', 'vasc':
  'Vascular lesions', 'df':
  'Dermatofibroma'
}
tile_df = pd.read_csv(os.path.join(base_skin_dir,
'HAM10000_metadata.csv')) tile_df['path'] =
tile_df['image_id'].map(imageid_path_dict.get)
tile_df['cell_type'] = tile_df['dx'].map(lesion_type_dict.get)
tile_df['cell_type_idx'] = pd.Categorical(tile_df['cell_type']).codes
tile_df.sample(3)
tile_df.describe(exclude=[np.number])
fig, ax1 = plt.subplots(1, 1, figsize = (10, 5))
tile_df['cell_type'].value_counts().plot(kind='bar', ax=ax1)
```



```
# load in all of the images
from skimage.io import imread
```

```
tile_df['image'] =

tile_df['path'].map(imread) # see the

image size distribution

tile_df['image'].map(lambda

x:x.shape).value_counts() (450,600, 3)  10015

Name: image, dtype:

int64 n_samples = 5

fig, m_axs = plt.subplots(7, n_samples, figsize = (4*n_samples,

3*7)) for n_axs, (type_name, type_rows) in zip(m_axs,

                          tile_df.sort_values(['cell_type']).groupby('cell_type')):

  n_axs[0].set_title(type_name)

  for c_ax, (_, c_row) in zip(n_axs,type_rows.sample(n_samples,
random_state=2018).iterrows()):

    c_ax.imshow(c_row['image'])

    c_ax.axis('off')

fig.savefig('category_samples.png', dpi=300)

rgb_info_df = tile_df.apply(lambda x: pd.Series({'{}_mean'.format(k): v
for k, v in

                      zip(['Red', 'Green', 'Blue'],

                        np.mean(x['image'], (0, 1)))}),1)

gray_col_vec = rgb_info_df.apply(lambda x:

np.mean(x), 1) for c_col in rgb_info_df.columns:

  rgb_info_df[c_col] =

rgb_info_df[c_col]/gray_col_vec

rgb_info_df['Gray_mean'] = gray_col_vec

rgb_info_df.sample(3)

for c_col in rgb_info_df.columns:

  tile_df[c_col] = rgb_info_df[c_col].values # we cant afford a copy
```

```
sns.pairplot(tile_df[['Red_mean', 'Green_mean', 'Blue_mean',
'Gray_mean', 'cell_type']],
          hue='cell_type', plot_kws = {'alpha': 0.5})
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional
indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the
future this will be interpreted as an array index, `arr[np.array(seq)]`,
which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

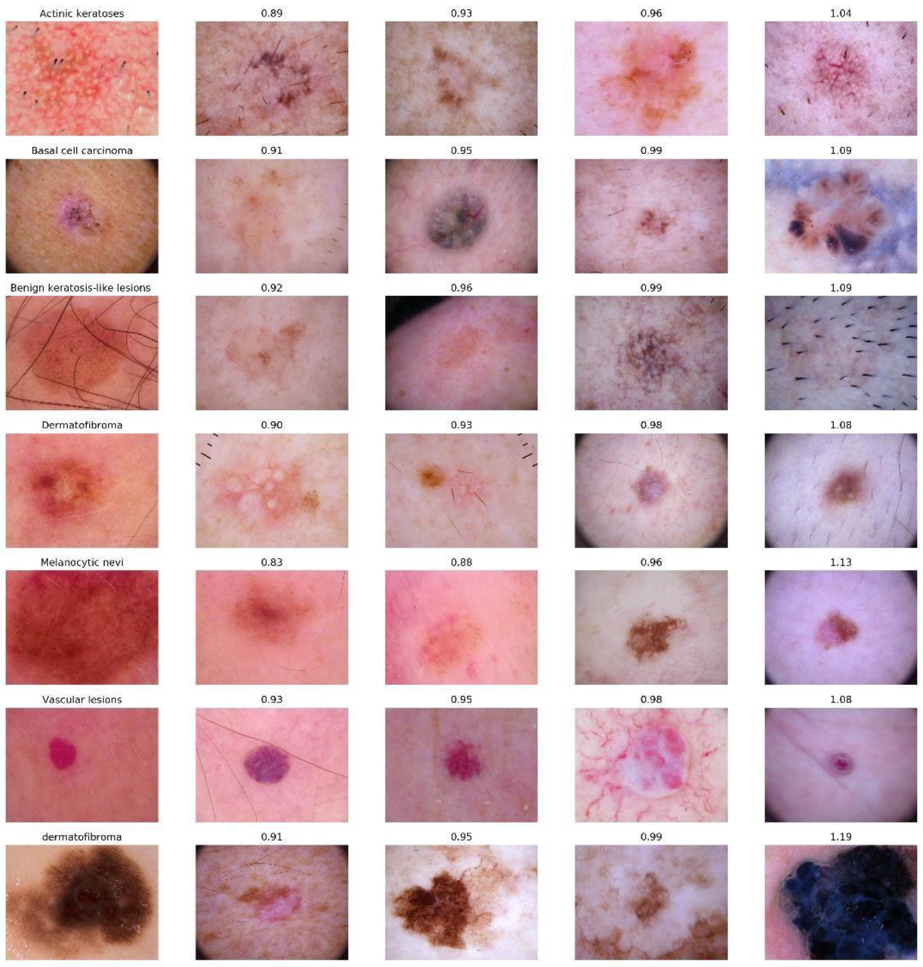
```
n_samples = 5
for sample_col in ['Red_mean', 'Green_mean', 'Blue_mean',
   'Gray_mean']: fig, m_axs = plt.subplots(7, n_samples, figsize =
   (4*n_samples, 3*7))
   def take_n_space(in_rows, val_col, n):
      s_rows =
      in_rows.sort_values([val_col])
      s_idx = np.linspace(0, s_rows.shape[0]-1, n,
      dtype=int) return s_rows.iloc[s_idx]
   for n_axs, (type_name, type_rows) in zip(m_axs,
                              tile_df.sort_values(['cell_type']).groupby('cell_type')):

      for c_ax, (_, c_row) in zip(n_axs,
                        take_n_space(type_rows,
                                 sample_col,
                                 n_samples).iterrows()):
         c_ax.imshow(c_row['image'])
         c_ax.axis('off')
         c_ax.set_title('{:2.2f}'.format(c_row[sample_col]))
      n_axs[0].set_title(type_name)
   fig.savefig('{}_samples.png'.format(sample_col), dpi=300)
```

```
from skimage.util import montage
rgb_stack = np.stack(tile_df.\
            sort_values(['cell_type', 'Red_mean'])['image'].\
            map(lambda x: x[::5, ::5]).values, 0)
rgb_montage = np.stack([montage(rgb_stack[:, :, :, i]) for i in
range(rgb_stack.shape[3])], -1)
print(rgb_montage.shape)
fig, ax1 = plt.subplots(1, 1, figsize = (20, 20), dpi=300)
ax1.imshow(rgb_montage)
fig.savefig('nice_montage.png')
from skimage.io import imsave
tile_df[['cell_type_idx',
'cell_type']].sort_values('cell_type_idx').drop_duplicate
s() from PIL import Image
def package_mnist_df(in_rows,
            image_col_name = 'image',
            label_col_name =
            'cell_type_idx',
            image_shape=(28, 28),
            image_mode='RGB',
            label_first=False
           ):
  out_vec_list = in_rows[image_col_name].map(lambda x:
                         np.array(Image.\
                              fromarray(x).
                              \
                              resize(image_shape,
resample=Image.LANCZOS).\
                              convert(image_mode)).ravel())
```

```
    out_vec = np.stack(out_vec_list, 0)
    out_df = pd.DataFrame(out_vec)
    n_col_names = ['pixel{:04d}'.format(i) for i in range(out_vec.shape[1])]
    out_df.columns = n_col_names
    out_df['label'] =
    in_rows[label_col_name].values.copy() if label_first:
        return
    out_df[['label']+n_col_names] else:
        return out_df
from itertools import product
for img_side_dim, img_mode in product([8, 28, 64, 128], ['L',
    'RGB']): if (img_side_dim==128) and (img_mode=='RGB'):
        # 128x128xRGB is a
        biggie break
    out_df = package_mnist_df(tile_df,
                          image_shape=(img_side_dim,
                         img_side_dim), image_mode=img_mode)
    out_path = f'hmnist_{img_side_dim}_{img_side_dim}_{img_mode}.csv'
    out_df.to_csv(out_path, index=False)
    print(f'Saved {out_df.shape} -> {out_path}:
{os.stat(out_path).st_size/1024:2.1f}kb')
```

OUTPUT:



Conclusion

Loading and preprocessing data is an essential step in any machine learning workflow. By following the steps outlined in this abstract, you can develop a general-purpose loading and preprocessing method that can be used for a variety of data types and tasks.