Лабораторная работа №1. Логистическая регрессия в качестве нейронной сети

```
import os
try:
  import wget
except:
  !pip install wget
  import wget
import tarfile


out_dir = 'data/not_mnist'
small_arhive = f'{out_dir}/notMNIST_small.tar.gz'
large_arhive = f'{out_dir}/notMNIST_large.tar.gz'
large_url = 'https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz'
small_url = 'https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz'


if not os.path.exists(out_dir):
  os.makedirs(out_dir)

if not os.path.exists(small_arhive):
  print(f"Downloading {small_arhive}.")
  wget.download(small_url, small_arhive)
  print()
else:
  print(f"Skipping {small_arhive} download (already exists)")

if not os.path.exists(large_arhive):
  print(f"Downloading {large_arhive}.")
  wget.download(large_url, large_arhive)
  print()
else:
  print(f"Skipping {large_arhive} download (already exists)")
```

```
Skipping data/not_mnist/notMNIST_small.tar.gz download (already exists)
Skipping data/not_mnist/notMNIST_large.tar.gz download (already exists)
```

```
print(f"Extracting {small_arhive}")
with tarfile.open(small_arhive) as tar:
  tar.extractall(out_dir)

print(f"Extracting {large_arhive}")
with tarfile.open(large_arhive) as tar:
  tar.extractall(out_dir)
```

```
Extracting data/not_mnist/notMNIST_small.tar.gz
Extracting data/not_mnist/notMNIST_large.tar.gz
```

```
import numpy as np
from pathlib import Path
from PIL import Image

def remove_duplicates(img_train, labels_train, img_test):
    img_new, labels_new = [], []
    test_set = {e.tostring() for e in img_test}
    for i, (x, y) in enumerate(zip(img_train, labels_train)):
        if x.tostring() not in test_set:
            img_new.append(x)
            labels_new.append(y)

    print(f'Removed {img_train.shape[0] - len(img_new)} duplicated images')
    return np.array(img_new), np.array(labels_new)

def load_images(path, n):
    labels = ['I', 'G', 'A', 'F', 'H', 'J', 'C', 'D', 'E', 'B']

    x, y = [], []
    for i, l in enumerate(labels):
        d = Path(path) / l
        print(f'Loading {str(d)} ', end='')
```

```
            for j, f in zip(range(n), d.iterdir()):
                try:
                    with Image.open(f) as img:
                        x.append(np.array(img))
                        y.append(i)
                except OSError:
                    pass
                if j % 1000 == 0:
                    print('.', end='', flush=True)
            print(flush=True)
        return np.array(labels), np.array(x), np.array(y)

    def load_not_mnist_data(path='data/not_mnist/', use_cache=True):
        train_folder = Path(path) / 'notMNIST_large'
        test_folder = Path(path) / 'notMNIST_small'

        train_cache_file = Path(path) / 'train.npz'
        test_cache_file = Path(path) / 'test.npz'

        if train_cache_file.exists() and test_cache_file.exists() and use_cache:
            f = np.load(train_cache_file)
            labels, img_train, labels_train = [v for k, v in f.items()]
            f = np.load(test_cache_file)
            labels, img_test, labels_test = [v for k, v in f.items()]
            print('Loaded cached arrays')

        else:
            labels, img_train, labels_train = load_images(train_folder, 10000000)
            labels, img_test, labels_test = load_images(test_folder, 10000000)
            np.savez(train_cache_file, labels, img_train, labels_train)
            np.savez(test_cache_file, labels, img_test, labels_test)

        return labels, img_train, labels_train, img_test, labels_test


    load_not_mnist_data()
```

```
Loaded cached arrays
(array(['I', 'G', 'A', 'F', 'H', 'J', 'C', 'D', 'E', 'B'], dtype='<U1'),
 array([[[180, 185, 214, ..., 217, 189, 184],
         [ 56,  58,  70, ...,  76,  66,  64],
         [  0,   0,   0, ...,   0,   0,   0],
         ...,
         [  0,   0,   0, ...,   0,   0,   0],
         [ 58,  60,  70, ...,  73,  64,  62],
         [177, 182, 211, ..., 209, 180, 174]],

        [[255, 255, 255, ...,  54,  48,  48],
         [255, 255, 255, ...,  54,  48,  48],
         [255, 255, 255, ...,  54,  48,  48],
         ...,
         [255, 255, 255, ...,  54,  48,  48],
         [255, 255, 255, ...,  54,  48,  48],
         [255, 255, 255, ...,  54,  48,  48]],

        [[  0,   0,   0, ...,   0,   0,   0],
         [  0,   0,   0, ...,   0,   0,   0],
         [  0,   0,   0, ...,   0,   0,   0],
         ...,
         [  0,   0,   0, ...,   0,   1,   0],
         [  0,   0,   0, ...,   0,   0,   0],
         [  0,   0,   0, ...,   0,   0,   0]],

        ...,

        [[  0,   0,   0, ...,   1,   0,   0],
         [  0,   0,   0, ...,   0,   1,   0],
         [  0,   0,   0, ...,   0,   3,   0],
         ...,
         [  0,   0,   0, ...,   0,   0,   0],
         [  0,   0,   0, ...,   0,   0,   0],
         [  0,   0,   0, ...,   0,   0,   0]],

        [[164, 175, 185, ...,   0,   0,   0],
         [255, 255, 255, ...,   0,   0,   0],
         [243, 253, 252, ...,   0,   0,   0],
         ...,
         [241, 255, 254, ...,   2,   1,   0],
         [241, 255, 254, ...,   1,   0,   0],
```

```
        [239, 250, 245, ...,   0,   0,   0]],

       [[  0,   0,   0, ...,  27,   0,   2],
        [  0,   0,   0, ..., 234,  68,   0],
        [  0,   0,   0, ..., 255, 232,  31],
        ...,
        [ 30,  91,  89, ...,   2,   0,   0],
        [128, 255, 253, ...,   0,   0,   0],
        [176, 255, 249, ...,   0,   0,   0]]], dtype=uint8),
 array([0, 0, 0, ..., 9, 9, 9]),
 array([[[ 81,  96, 138, ..., 124,  60,  37],
        [131, 153, 215, ..., 191,  94,  59],
        [122, 145, 210, ..., 188,  90,  55],
        ...,
        [124, 147, 212, ..., 190,  92,  56],
        [125, 148, 213, ..., 190,  92,  56],
        [120, 143, 206, ..., 184,  89,  54]],

       [[  0,   0,   0, ..., 134, 150, 110],
        [  0,   0,   0, ..., 255, 145,  40],
        [  0,   0,   0, ..., 132,   0,   0],
        ...,
        [100, 255, 133, ...,   0,   0,   0],
        [ 11, 172, 238, ...,   0,   0,   0],
        [  0,   0,  98, ...,   0,   0,   0]],

       [[  8,  16,  23, ..., 107, 116, 119],
        [176, 170, 166, ..., 244, 251, 255],
        [211, 236, 255, ..., 255, 254, 253],
        ...,
        [202, 233, 255, ..., 255, 255, 255],
        [212, 233, 247, ..., 227, 237, 244],
        [110,  98,  89, ...,  71,  62,  48]],

       ...,

       [[ 13,   0,   6, ...,   0,   0,   0],
        [236, 189, 216, ...,   0,   1,   0],
        [190, 255, 255, ...,   2,   0,   0],
        ...,
        [159, 249, 255, ...,   0,   2,   0],
        [125, 161, 239, ...,   0,   0,   0],
```

```
          [  0,  17, 114, ...,   0,   0,   0]],

         [[  0,   0,   0, ...,   0,   2,   0],
          [  0,   0,   0, ..., 136,   2,   0],
          [  0,   0,   0, ..., 255, 143,   0],
          ...,
          [  0,   0,   3, ...,   2,   0,   0],
          [  0,   1,   2, ...,   0,   0,   0],
          [137, 210, 217, ...,   0,   0,   0]],

         [[217, 224, 246, ...,   0,   0,   0],
          [  4,  13,  42, ...,   3,   0,   0],
          [  0,   1,   0, ...,   0,   2,   0],
          ...,
          [  0,   1,   0, ...,  80,   0,   3],
          [  6,  20,  55, ...,   0,   3,   0],
          [219, 229, 252, ...,   3,   0,   0]]], dtype=uint8),
 array([0, 0, 0, ..., 9, 9, 9]))
```

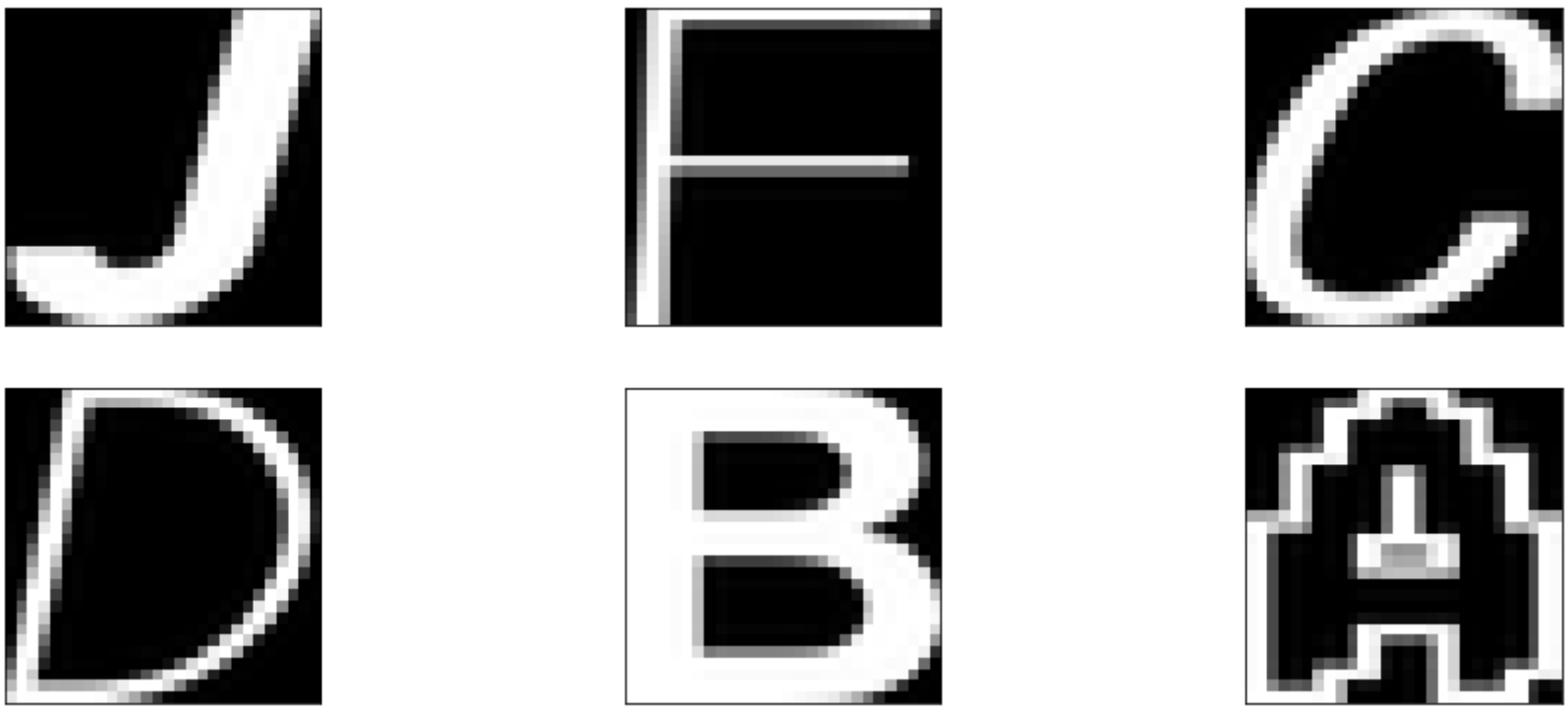
```
labels, img_train, labels_train, img_test, labels_test = load_not_mnist_data()
```

```
Loaded cached arrays
```

Задание 1. Загрузите данные и отобразите на экране несколько из изображений с помощью языка Python;

```
import matplotlib.pyplot as plt

plt.rcParams['font.size'] = 14
plt.rcParams['font.family'] = 'consolas'

rows = 2
cols = 3
fig = plt.figure(figsize=(16, 6.5))
for i in range(1, cols * rows + 1):
    ax = fig.add_subplot(rows, cols, i)
    ax.set_xticks([])
    ax.set_yticks([])
```

```
    j = np.random.randint(0, labels_test.shape[0] - 1)
    plt.imshow(img_test[j], cmap='gray')
```

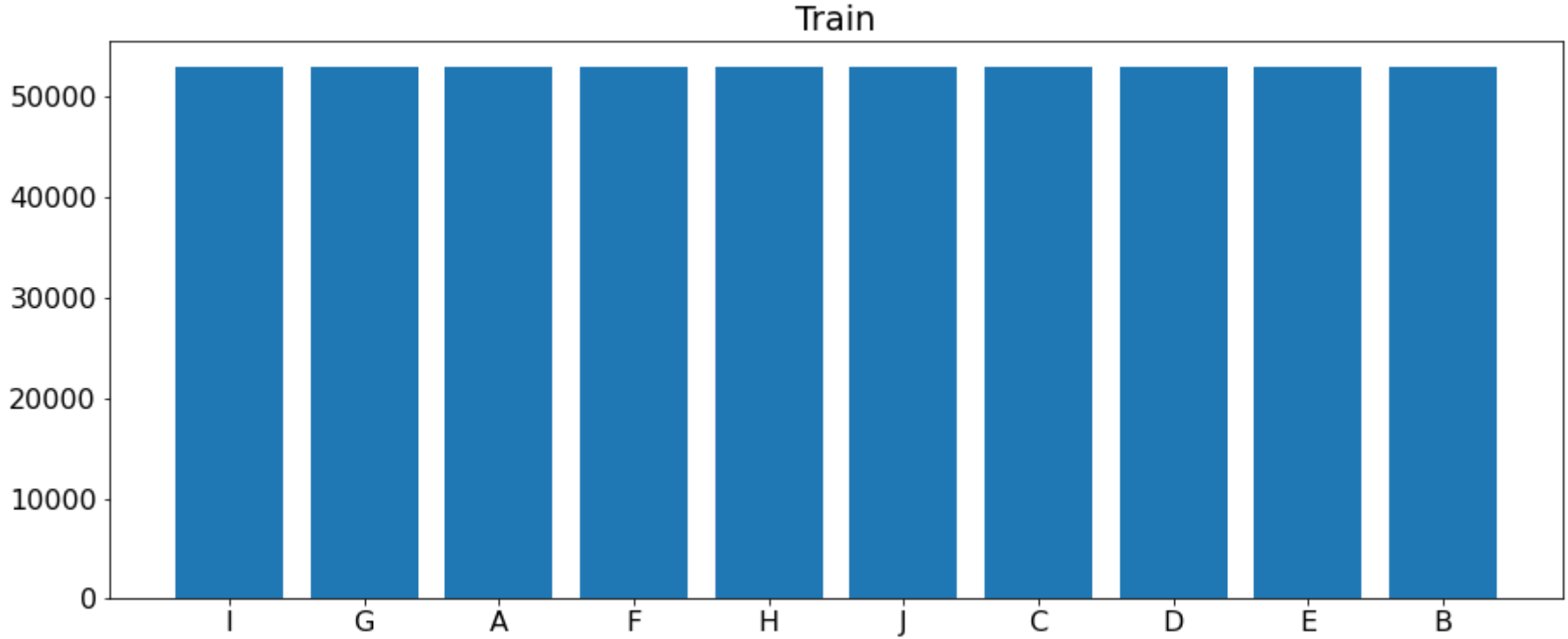


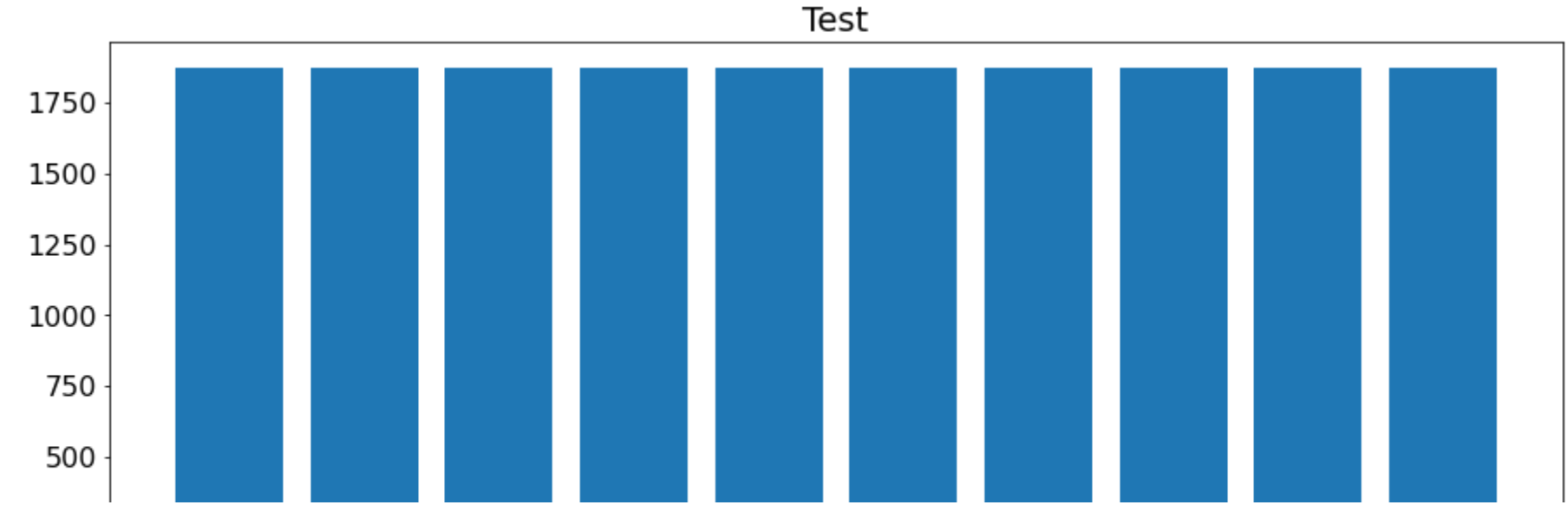
Задание 2. Проверьте, что классы являются сбалансированными, т.е. количество изображений, принадлежащих каждому из классов, примерно одинаково (В данной задаче 10 классов).

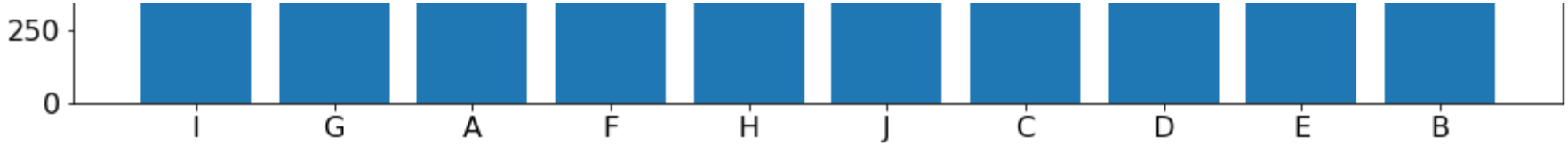
```
from collections import defaultdict

plt.rcParams['font.size'] = 16

def show_balance(title, ax, a, labels):
    counts = defaultdict(int)
    for e in a:
        counts[labels[e]] += 1

    print(counts)
```

```
        print(counts)

        ax.bar(counts.keys(), counts.values())
        ax.set_title(title)

    fig = plt.figure(figsize=(15, 13), )
    show_balance("Train", fig.add_subplot(2, 1, 1), labels_train, labels)
    show_balance("Test", fig.add_subplot(2, 1, 2), labels_test, labels)
```

```
defaultdict(<class 'int'>, {'I': 52912, 'G': 52912, 'A': 52909, 'F': 52912, 'H': 52912, 'J': 52911, 'C': 52912, 'D': 5
defaultdict(<class 'int'>, {'I': 1872, 'G': 1872, 'A': 1872, 'F': 1872, 'H': 1872, 'J': 1872, 'C': 1873, 'D': 1873, 'E
```







Задание 3,4 Разделение на обучающую, валидационную и контрольную выборки. Удаление дубликатов

```
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.model_selection import train_test_split

def flatten(a):
    return a.reshape(a.shape[0], a.shape[1] * a.shape[2])

def load_data():
    labels, img_train, labels_train, img_test, labels_test = load_not_mnist_data()
    img_train, labels_train = remove_duplicates(img_train, labels_train, img_test)
    return labels, flatten(img_train), labels_train, flatten(img_test), labels_test



_, x_train, y_train, x_test, y_test = load_data()
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.1)
```

```
Loaded cached arrays
Removed 12213 duplicated images
```

Задание 5. Постройте простейший классификатор (например, с помощью логистической регрессии). Постройте график зависимости точности классификатора от размера обучающей выборки (50, 100, 1000, 50000)

```
results = {}
results.setdefault('val_acc', {})
ns = [50, 100, 1000, 50000]
```

```
print('Training Logistic Regression model:')
for n in ns:
  indices = np.arange(len(x_train))
  if n < len(x_train):
    indices = np.random.choice(indices, n, replace=False)

  # model = SGDClassifier(loss='log', tol=1e-4, early_stopping=True)
  model = LogisticRegression(solver='lbfgs', max_iter = 1000000)
  model.fit(x_train[indices], y_train[indices])

  y_pred = model.predict(x_val)
  results["val_acc"][str(n)] = acc = np.mean(np.equal(y_pred, y_val).astype(np.int))
  print(f"n = {n}, accuracy = {acc:.5f}, iterations = {model.n_iter_}")
```

```
Training Logistic Regression model:
n = 50, accuracy = 0.56613, iterations = [60]
n = 100, accuracy = 0.69766, iterations = [97]
n = 1000, accuracy = 0.75181, iterations = [486]
n = 50000, accuracy = 0.81211, iterations = [4805]
```

```
print (results["val_acc"].keys())
print (results["val_acc"].values())
```

```
dict_keys(['50', '100', '1000', '50000'])
dict_values([0.566133369445358, 0.6976649706912228, 0.7518136619527578, 0.8121142945580468])
```

```
# The Data
x = list(results["val_acc"].keys())
y = list(results["val_acc"].values())

# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Logistic Regression')

# Plot the data
ax.plot(x,y)

# Show the grid lines as dark grey lines
```

```
# Show the grid lines as dark grey lines
plt.grid(b=True, which='major', color='#666666', linestyle='-')

plt.show()
```