```python
In [1]: #import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch.utils.data import random_split
from torch.utils.data import DataLoader,Dataset
import torch.nn.functional as F
from torch.utils.tensorboard import SummaryWriter
from sklearn.metrics import r2_score
import cv2
import os
```

```python
In [2]: #getting the path to the folder
dir=os.getcwd()
```

```python
In [3]: #paths to data
albedo=dir+"/DATA/Data_Albedo/Albedo_Map.csv"
LPFe_Map=dir+"/DATA/Data_Albedo/LPFe_Map.csv"
LPK_Map=dir+"/DATA/Data_Albedo/LPK_Map.csv"
LPTh_Map=dir+"/DATA/Data_Albedo/LPTh_Map.csv"
LPTi_Map=dir+"/DATA/Data_Albedo/LPTi_Map.csv"
```

```python
In [4]: #custom dataset for nearby pixels
class dataset_nearby_pixel(Dataset):
    def __init__(self,path_1,path_2,path_3,path_4,path_5,mode):
        self.path_1=path_1
        self.path_2=path_2
        self.path_3=path_3
        self.path_4=path_4
        self.path_5=path_5
        self.mode=mode
        #converting the data to ndarray
        self.X_1=np.array(pd.read_csv(path_1))
        self.X_2=np.array(pd.read_csv(path_2))
        self.X_3=np.array(pd.read_csv(path_3))
        self.X_4=np.array(pd.read_csv(path_4))
        self.X_5=cv2.GaussianBlur(np.array(pd.read_csv(path_5)),ksize=(0,0),sigmaX=9)
        self.X=[]
        self.Y=[]
        n,m=np.shape(self.X_5)[0],np.shape(self.X_5)[1]
        for i in range(1,n-1):
            for j in range(1,m-1):
                nearby=[]
                for chem in [self.X_1,self.X_2,self.X_3,self.X_4]:
                    for a in [-1,0,1]:
                        for b in [-1,0,1]:
                            nearby.append(chem[i+a][j+b])
                self.X.append(nearby)
                self.Y.append([self.X_5[i][j]])
        l=len(self.Y)//2
        if mode == "Train" or "train":
            self.X=np.array(self.X)[:l]
            self.Y=np.array(self.Y)[:l]
        elif mode== "Test" or "test":
            self.X=np.array(self.X)[l:]
            self.Y=np.array(self.Y)[l:]
    def __len__(self):
        self.filelength=np.shape(self.Y)[0]
        return self.filelength

    def __getitem__(self,idx):
        return torch.from_numpy(self.X[idx]),torch.from_numpy(self.Y[idx])
```

```python
In [5]: class autoencoder(torch.nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        # Encoder Network
        self.encoder = torch.nn.Sequential(torch.nn.Linear(36,72),
                                           torch.nn.ReLU(True),
                                           torch.nn.Linear(72,148))
        # Decoder Network
        self.decoder = torch.nn.Sequential(torch.nn.Linear(148,72),
                                           torch.nn.ReLU(True),
                                           torch.nn.Linear(72, 36),
                                           torch.nn.ReLU(True),
                                           torch.nn.Linear(36,1),
                                           torch.nn.ReLU(True))

    def forward(self,x):
        x=self.encoder(x)
        x=self.decoder(x)
        return x
```

```python
In [6]: class train():
    def __init__(self,batch_size,epochs,lr,train_val_split,scheduler,near):
        self.batch_size=batch_size
        self.scheduler=scheduler
        self.epochs=epochs
        self.lr=lr
        self.train_val_split=train_val_split
        self.near=near
        if self.near== True:
            self.data=dataset_nearby_pixel(LPFe_Map,LPK_Map,LPTh_Map,LPTi_Map,albedo,mode="train")
            self.train_data,self.val_data=random_split(self.data,[len(self.data)-int(self.train_val_split*len(s
            self.train_loader=DataLoader(self.train_data,batch_size=self.batch_size,shuffle=True)
            self.val_loader=DataLoader(self.val_data,batch_size=self.batch_size,shuffle=True)
            self.net=autoencoder()
        # else:
        #     self.data=dataset(LPFe_Map,LPK_Map,LPTh_Map,LPTi_Map,albedo,0,360)
        #     self.train_data,self.val_data=random_split(self.data,[len(self.data)-int(self.train_val_split*len
        #     self.train_loader=DataLoader(self.train_data,batch_size=self.batch_size,shuffle=True)
        #     self.val_loader=DataLoader(self.val_data,batch_size=self.batch_size,shuffle=True)
        #     self.net=model(n_feature=4, n_hidden=4, n_output=1)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=self.lr)
        if self.scheduler==True:
            self.sched=torch.optim.lr_scheduler.ExponentialLR(self.optimizer,gamma=0.7)
        self.loss_func = torch.nn.MSELoss()
        self.writer = SummaryWriter()

    def trainer(self):
        self.net=self.net.train()
        self.net=self.net.cuda()
        for epoch in range(self.epochs):
            for input,gt in self.train_loader:
                input = input.cuda()
                gt = gt.cuda()
                gt=torch.reshape(gt,(len(gt),1))
                output = self.net(input.float())
                loss = self.loss_func(output, gt.float())
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()
            if self.scheduler == True:
                self.sched.step()
            print('Epoch : {},  train loss : {}'.format(epoch+1,loss.item()))
            with torch.no_grad():
                for input,gt in self.val_loader:
                    input=input.cuda()
                    gt= gt.cuda()
                    gt=torch.reshape(gt,(len(gt),1))
                    val_output = self.net(input.float())
                    val_loss = self.loss_func(val_output,gt.float())
            print('Epoch : {},  val_loss : {}'.format(epoch+1,val_loss.item()))
            self.writer.add_scalar("Loss/train", loss, epoch)
            self.writer.add_scalar("Loss/val", val_loss, epoch)
            if self.scheduler == True:
                self.writer.add_scalar("lr/epoch",self.lr,epoch)
        torch.save(self.net.state_dict(),f"albedo_autoencoder_{self.epochs}_{self.lr}_{self.batch_size}.pth")
```

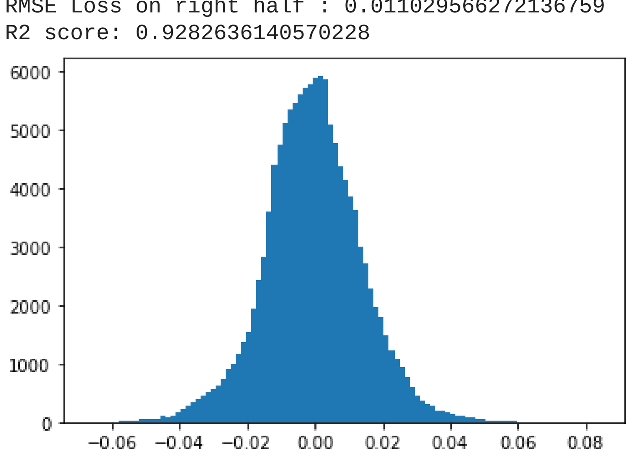
```python
In [7]: train_best_near=train(batch_size=64,epochs=100,lr=0.0001,train_val_split=0.3,scheduler=False,near=True)
train_best_near.trainer()
```

```
2022-04-10 16:56:31.090501: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dyna
mic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: /home/g0kul6/miniconda3/lib/python3.9/site-packages/cv2/../../lib64:
2022-04-10 16:56:31.090532: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if
you do not have a GPU set up on your machine.
Epoch : 1,  train loss : 0.0878107100725174
Epoch : 1,  val_loss : 0.09150587022304535
Epoch : 2,  train loss : 0.08635705709457397
Epoch : 2,  val_loss : 0.08455988019704819
Epoch : 3,  train loss : 0.08399862796068192
Epoch : 3,  val_loss : 0.09489002823829651
Epoch : 4,  train loss : 0.08470350503921509
Epoch : 4,  val_loss : 0.0816386342048645
Epoch : 5,  train loss : 0.07871576398611069
Epoch : 5,  val_loss : 0.07830406725406647
Epoch : 6,  train loss : 0.08545029908418655
Epoch : 6,  val_loss : 0.08926884829998016
Epoch : 7,  train loss : 0.07614102959632874
Epoch : 7,  val_loss : 0.08242516964673996
Epoch : 8,  train loss : 0.001455507823266089
Epoch : 8,  val_loss : 0.0037453914992511272
Epoch : 9,  train loss : 0.002118888543918729
Epoch : 9,  val_loss : 0.0018569575622677803
Epoch : 10,  train loss : 0.000737999624107033
Epoch : 10,  val_loss : 0.0023089596070349216
Epoch : 11,  train loss : 0.0009930769447237253
Epoch : 11,  val_loss : 0.0012720798840746284
Epoch : 12,  train loss : 0.00169272068887949
Epoch : 12,  val_loss : 0.003894631750881672
Epoch : 13,  train loss : 0.0007690248312428594
Epoch : 13,  val_loss : 0.0005912671331316233
Epoch : 14,  train loss : 0.0006027873605489731
Epoch : 14,  val_loss : 0.0033615445718169212
Epoch : 15,  train loss : 0.00039357933565042913
Epoch : 15,  val_loss : 0.001383837079629302
Epoch : 16,  train loss : 0.00035889967693947256
Epoch : 16,  val_loss : 0.0006209786515682936
Epoch : 17,  train loss : 0.0010290767531841993
Epoch : 17,  val_loss : 0.00023587308533024043
Epoch : 18,  train loss : 0.00041479113860987127
Epoch : 18,  val_loss : 0.0005015709903091192
Epoch : 19,  train loss : 0.0005539472913369536
Epoch : 19,  val_loss : 0.0004221520503051579
Epoch : 20,  train loss : 0.0004449983825907111
Epoch : 20,  val_loss : 0.0005475503276102245
Epoch : 21,  train loss : 0.00039325794205069540
Epoch : 21,  val_loss : 0.000538503285497427
Epoch : 22,  train loss : 0.0005059874383732677
Epoch : 22,  val_loss : 0.000414421025197953
Epoch : 23,  train loss : 0.00033786913263611495
Epoch : 23,  val_loss : 0.00037795904790982604
Epoch : 24,  train loss : 0.0005521064158529043
Epoch : 24,  val_loss : 0.0005237184232100844
Epoch : 25,  train loss : 0.00031337534892372787
Epoch : 25,  val_loss : 0.0003929006343241781
Epoch : 26,  train loss : 0.000631699978839606
Epoch : 26,  val_loss : 0.0003714418853633106
Epoch : 27,  train loss : 0.0004454639565665275
Epoch : 27,  val_loss : 0.0003284956910647452
Epoch : 28,  train loss : 0.0003302760305814445
Epoch : 28,  val_loss : 0.00032676703995093703
Epoch : 29,  train loss : 0.00031398964347317815
Epoch : 29,  val_loss : 0.0002015146892517805
Epoch : 30,  train loss : 0.0003893116081599146
Epoch : 30,  val_loss : 0.00039379074587486684
Epoch : 31,  train loss : 0.0002021557738771662
Epoch : 31,  val_loss : 0.0003710713644977659
Epoch : 32,  train loss : 0.00042482971912249923
Epoch : 32,  val_loss : 0.00040611831354908645
Epoch : 33,  train loss : 0.00035993478377349675
Epoch : 33,  val_loss : 0.00027809111634269357
Epoch : 34,  train loss : 0.0003080436435993761
Epoch : 34,  val_loss : 0.0004221129638608545
Epoch : 35,  train loss : 0.00034128897823393345
Epoch : 35,  val_loss : 0.00028266041772440076
Epoch : 36,  train loss : 0.0003469839575700462
Epoch : 36,  val_loss : 0.0002876578364521265
Epoch : 37,  train loss : 0.00029542899574153125
Epoch : 37,  val_loss : 0.00028885542997159064
Epoch : 38,  train loss : 0.00021320061932783574
Epoch : 38,  val_loss : 0.00031388140632770956
Epoch : 39,  train loss : 0.0002388346183579415
Epoch : 39,  val_loss : 0.0002876596408896148
Epoch : 40,  train loss : 0.00031169658177532256
Epoch : 40,  val_loss : 0.00019342736050020903
Epoch : 41,  train loss : 0.00036268055555410683
Epoch : 41,  val_loss : 0.0002429357700748369
Epoch : 42,  train loss : 0.0003934462438337505
Epoch : 42,  val_loss : 0.00018068814824800938
Epoch : 43,  train loss : 0.00029094802448526025
Epoch : 43,  val_loss : 0.00024218449834734201
Epoch : 44,  train loss : 0.00019266123126726598
Epoch : 44,  val_loss : 0.00022934206936042756
Epoch : 45,  train loss : 0.0002391851885477081
Epoch : 45,  val_loss : 0.0003555946168489754
Epoch : 46,  train loss : 0.0003869547217618674
Epoch : 46,  val_loss : 0.00028652395121753216
Epoch : 47,  train loss : 0.00025124201783910394
Epoch : 47,  val_loss : 0.000255148479482159
Epoch : 48,  train loss : 0.00028794832178391516
Epoch : 48,  val_loss : 0.00014857907081022859
Epoch : 49,  train loss : 0.00014865322737023234
Epoch : 49,  val_loss : 0.00018187488603871316
Epoch : 50,  train loss : 0.00023415272880811244
Epoch : 50,  val_loss : 0.0003235582262277603
Epoch : 51,  train loss : 0.0002520301495678723
Epoch : 51,  val_loss : 0.00019141320080962032
Epoch : 52,  train loss : 0.0002367224806221202
Epoch : 52,  val_loss : 0.00024819208192639053
Epoch : 53,  train loss : 0.00017335719894617796
Epoch : 53,  val_loss : 0.00022475211881101131
Epoch : 54,  train loss : 0.00018994287529494613
Epoch : 54,  val_loss : 0.00023951800540089607
Epoch : 55,  train loss : 0.0003508939116727561
Epoch : 55,  val_loss : 0.00020230821974109858
Epoch : 56,  train loss : 0.0003284876001998782
Epoch : 56,  val_loss : 0.000191724015166983
Epoch : 57,  train loss : 0.00027401631814427674
Epoch : 57,  val_loss : 0.00017758674221113324
Epoch : 58,  train loss : 0.00018351306789554656
Epoch : 58,  val_loss : 0.00027551455423235893
Epoch : 59,  train loss : 0.00029632699443027377
Epoch : 59,  val_loss : 0.00026373896980658174
Epoch : 60,  train loss : 0.00028729852056130767
Epoch : 60,  val_loss : 0.00023372776922769845
Epoch : 61,  train loss : 0.00016634882194921374
Epoch : 61,  val_loss : 0.00023168826010078192
Epoch : 62,  train loss : 0.00020327151287347078
Epoch : 62,  val_loss : 0.00021067260240670294
Epoch : 63,  train loss : 0.00019967790285591036
Epoch : 63,  val_loss : 0.00013012014096602798
Epoch : 64,  train loss : 0.0001605566794751212
Epoch : 64,  val_loss : 0.00034965912345796824
Epoch : 65,  train loss : 0.00018572727276477963
Epoch : 65,  val_loss : 0.00019503121438901871
Epoch : 66,  train loss : 0.0001818834716686979
Epoch : 66,  val_loss : 0.00017112800560425967
Epoch : 67,  train loss : 0.0003523861523717642
Epoch : 67,  val_loss : 0.0002301791391801089
Epoch : 68,  train loss : 0.0001988007134059444
Epoch : 68,  val_loss : 0.00015962244651746005
Epoch : 69,  train loss : 0.0002651807735674083
Epoch : 69,  val_loss : 0.00022040950716473162
Epoch : 70,  train loss : 0.0001922467927215621
Epoch : 70,  val_loss : 0.00017349841073155403
Epoch : 71,  train loss : 0.0001790209935279563
Epoch : 71,  val_loss : 0.00025164944236166775
Epoch : 72,  train loss : 0.0003164437075611204
Epoch : 72,  val_loss : 0.00028819861472584307
Epoch : 73,  train loss : 0.00026409985730424523
Epoch : 73,  val_loss : 0.0001858785399235785
Epoch : 74,  train loss : 0.0001594211789779365
Epoch : 74,  val_loss : 0.0002197386638727039
Epoch : 75,  train loss : 0.00031874972046352923
Epoch : 75,  val_loss : 0.00021966306667309254
Epoch : 76,  train loss : 0.00023409670393448323
Epoch : 76,  val_loss : 0.00023453112225979567
Epoch : 77,  train loss : 0.00021693574672099203
Epoch : 77,  val_loss : 0.0001339162845397368
Epoch : 78,  train loss : 0.00028131663566455245
Epoch : 78,  val_loss : 0.00014900673704687506
Epoch : 79,  train loss : 0.00013375085836742073
Epoch : 79,  val_loss : 0.00017710859538055956
Epoch : 80,  train loss : 0.0002089966437779367
Epoch : 80,  val_loss : 0.00027121807215735316
Epoch : 81,  train loss : 0.00011807464034063742
Epoch : 81,  val_loss : 0.0001936611661221832
Epoch : 82,  train loss : 0.00026545108994469047
Epoch : 82,  val_loss : 0.00018120225286111236
Epoch : 83,  train loss : 0.00030542834429070354
Epoch : 83,  val_loss : 0.00016551249427720904
Epoch : 84,  train loss : 0.00020824349485337734
Epoch : 84,  val_loss : 0.00017518011736683547
Epoch : 85,  train loss : 0.00016671160119585693
Epoch : 85,  val_loss : 0.0002496769593562931
Epoch : 86,  train loss : 0.00016189822054002434
Epoch : 86,  val_loss : 0.00023654475808143616
Epoch : 87,  train loss : 0.00017476428183726966
Epoch : 87,  val_loss : 0.00015450453793164343
Epoch : 88,  train loss : 0.00022608425933867693
Epoch : 88,  val_loss : 0.00024215469602495432
Epoch : 89,  train loss : 0.00023599425912834704
Epoch : 89,  val_loss : 0.00019877507293131202
Epoch : 90,  train loss : 0.0002211619139416143
Epoch : 90,  val_loss : 0.0002083896251861006
Epoch : 91,  train loss : 0.00024875058443285525
Epoch : 91,  val_loss : 0.00015036266995593905
Epoch : 92,  train loss : 0.00014178614947013557
Epoch : 92,  val_loss : 0.00020575242524500936
Epoch : 93,  train loss : 0.00023567077005282044
Epoch : 93,  val_loss : 0.00018904960597865283
Epoch : 94,  train loss : 0.00017278823361266404
Epoch : 94,  val_loss : 0.00020089151803404093
Epoch : 95,  train loss : 0.0001762991159921512
Epoch : 95,  val_loss : 0.00014999501581769437
Epoch : 96,  train loss : 0.000283252535155043
Epoch : 96,  val_loss : 0.0002580283035058528
Epoch : 97,  train loss : 0.0001948497083503753
Epoch : 97,  val_loss : 0.00035074364859610796
Epoch : 98,  train loss : 0.00019196790526621044
Epoch : 98,  val_loss : 0.00021482181909959763
Epoch : 99,  train loss : 0.00023469347797799855
Epoch : 99,  val_loss : 0.00019949045963585377
Epoch : 100,  train loss : 0.00022595925838686526
Epoch : 100,  val_loss : 0.00027036419487558305
```

```python
In [7]: test_data_nearby=dataset_nearby_pixel(LPFe_Map,LPK_Map,LPTh_Map,LPTi_Map,albedo,mode="test")
test_loader_nearby=DataLoader(test_data_nearby,batch_size=1)
loss_function=torch.nn.MSELoss()
net_test=autoencoder()
net_test=net_test.cuda()
net_test.load_state_dict(torch.load(dir+"/albedo_autoencoder_100_0.0001_64.pth"))
right_predicted_near=[]
right_truth_near=[]
total_loss_near=[]
net_test=net_test.eval()
for i,l in test_loader_nearby:
    i=i.cuda()
    l=l.cuda()
    l=torch.reshape(l,(len(l),1))
    output=net_test(i.float())
    loss=loss_function(output,l.float())
    loss=loss.cpu().item()
    total_loss_near.append(np.sqrt(loss))
    right_predicted_near.append(output.cpu().item())
    right_truth_near.append(l.cpu().item())


print("RMSE Loss on right half :",np.mean(total_loss_near))
print("R2 score:",r2_score(right_truth_near,right_predicted_near))
residual_near=np.subtract(right_predicted_near,right_truth_near)
plt.hist(residual_near,bins=100)
plt.show()
```

```
RMSE Loss on right half : 0.011029566272136759
R2 score: 0.9282636140570228
```

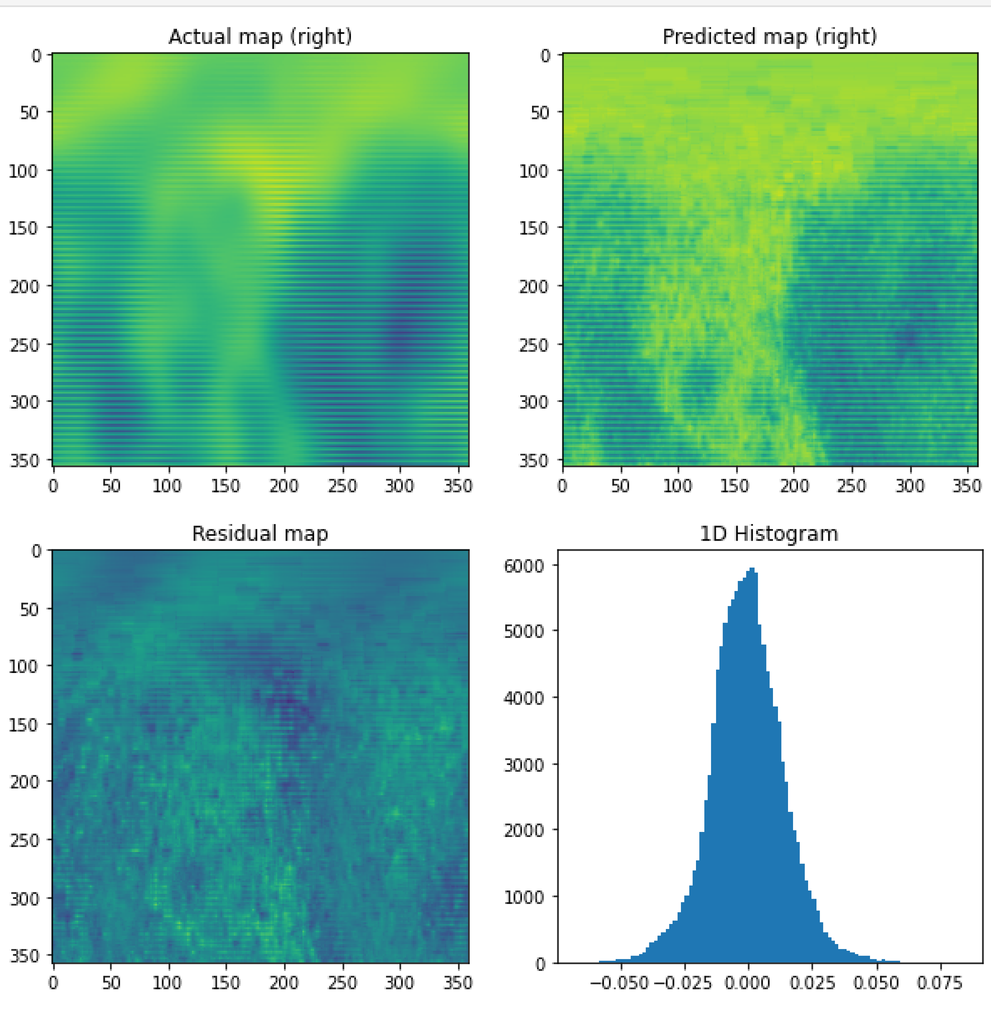


```python
In [10]: ffig, faxes = plt.subplots(2, 2, figsize=(10, 10))
faxes[0, 0].imshow(np.reshape(right_truth_near,(357,359)))
faxes[0, 0].set_title("Actual map (right)")
faxes[0, 0].grid(False)

faxes[0, 1].imshow(np.reshape(right_predicted_near,(357,359)))
faxes[0, 1].set_title("Predicted map (right)")
faxes[0, 1].grid(False)

faxes[1, 0].imshow(np.reshape(residual_near,(357,359)))
faxes[1, 0].set_title("Residual map")
faxes[1, 0].grid(False)

faxes[1, 1].hist(residual_near, bins=100)
faxes[1, 1].set_title("1D Histogram")
faxes[1, 1].grid(False)
```




```python
In [ ]:
```