# Brain Tumor Radiogenomic Classification - EDA and Modeling

Predict the status of a genetic biomarker important for brain cancer treatment

## Import Libraries

In [1]:
```
import os
import json
import glob
import random
import collections

import numpy as np
import pandas as pd
import pydicom
from pydicom.pixel_data_handlers.util import apply_voi_lut
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
```

**train/** - folder containing the training files, with each top-level folder representing a subject
**train_labels.csv** - file containing the target MGMT_value for each subject in the training data (e.g. the presence of MGMT promoter methylation)
**test/** - the test files, which use the same structure as train/; your task is to predict the MGMT_value for each subject in the test data.

## Read data

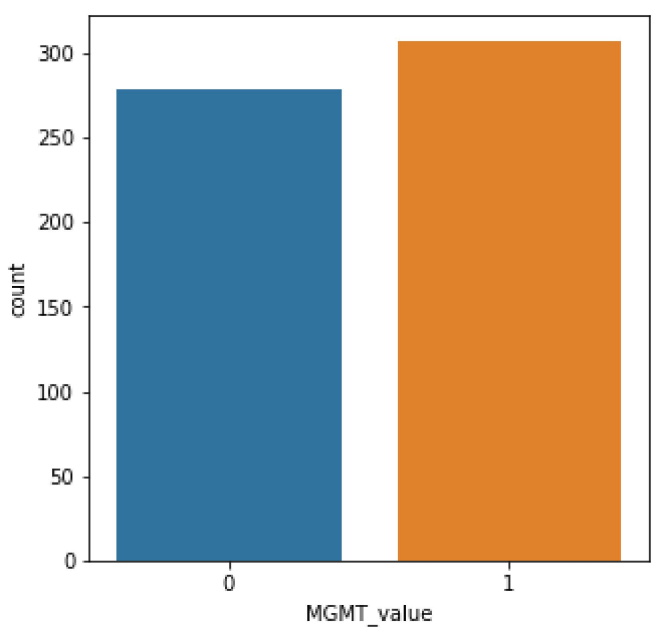
In [2]:
```
train_df = pd.read_csv("../input/rsna-miccai-brain-tumor-radiogenomic-classification/train_labels.csv")
train_df
```

Out[2]:

| | BraTS21ID | MGMT_value |
|---|---|---|
| **0** | 0 | 1 |
| **1** | 2 | 1 |
| **2** | 3 | 0 |
| **3** | 5 | 1 |
| **4** | 6 | 1 |
| **...** | ... | ... |
| **580** | 1005 | 1 |
| **581** | 1007 | 1 |
| **582** | 1008 | 1 |
| **583** | 1009 | 0 |
| **584** | 1010 | 0 |

585 rows × 2 columns

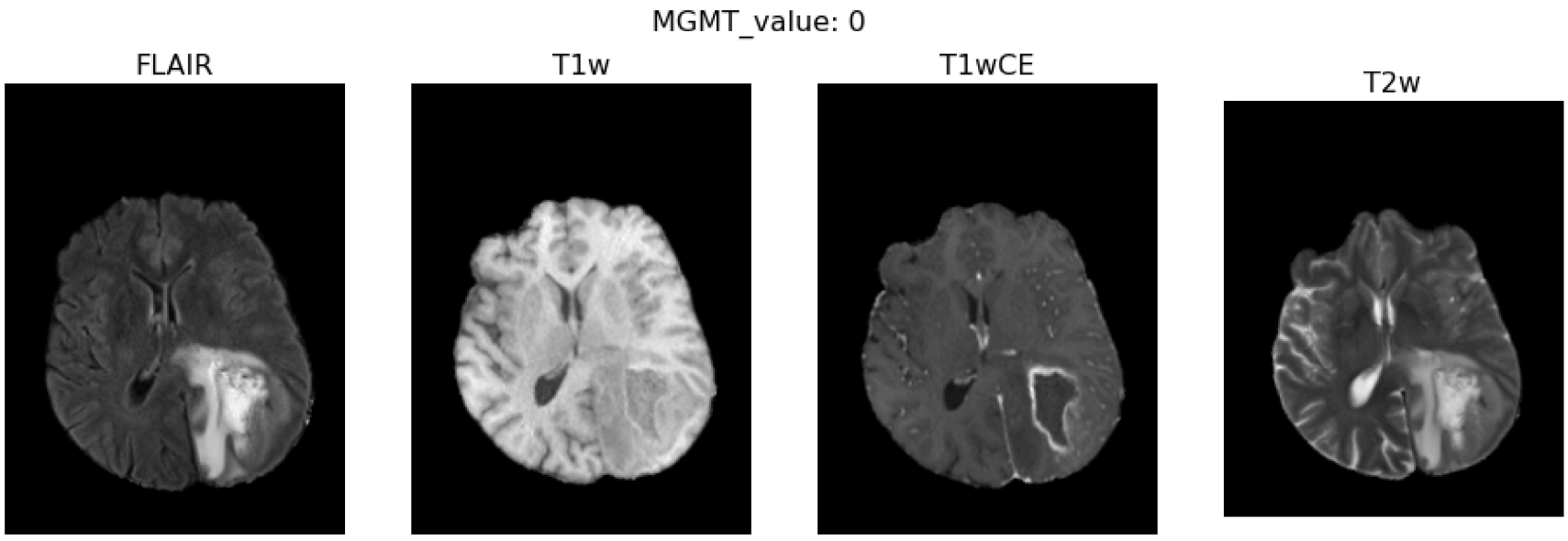
In [3]:
```
plt.figure(figsize=(5, 5))
sns.countplot(data=train_df, x="MGMT_value");
```

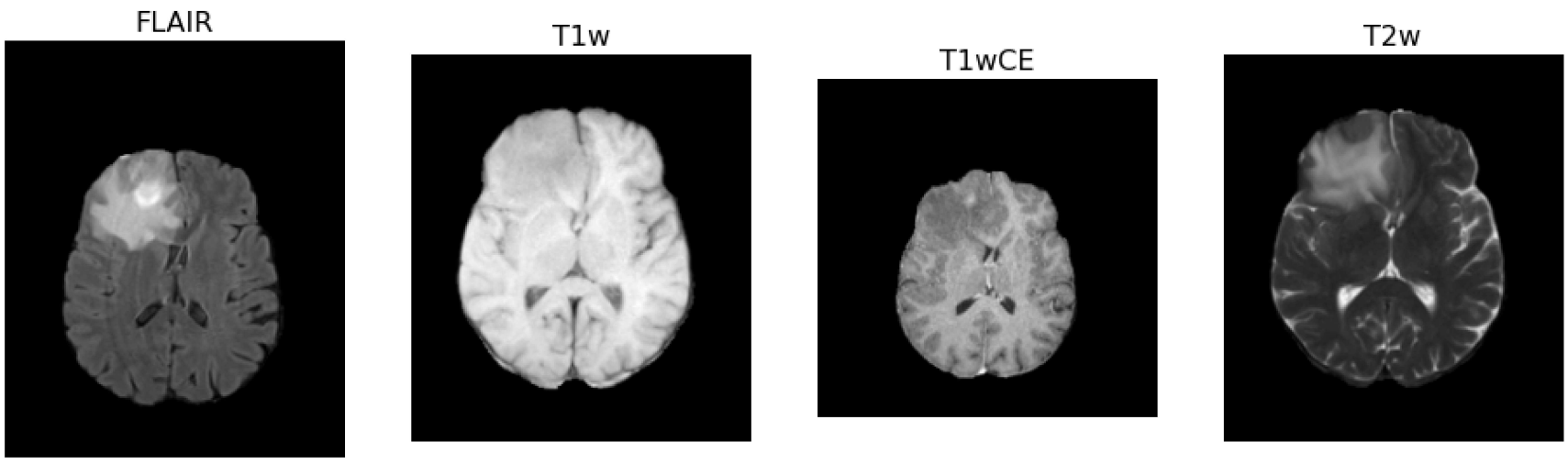


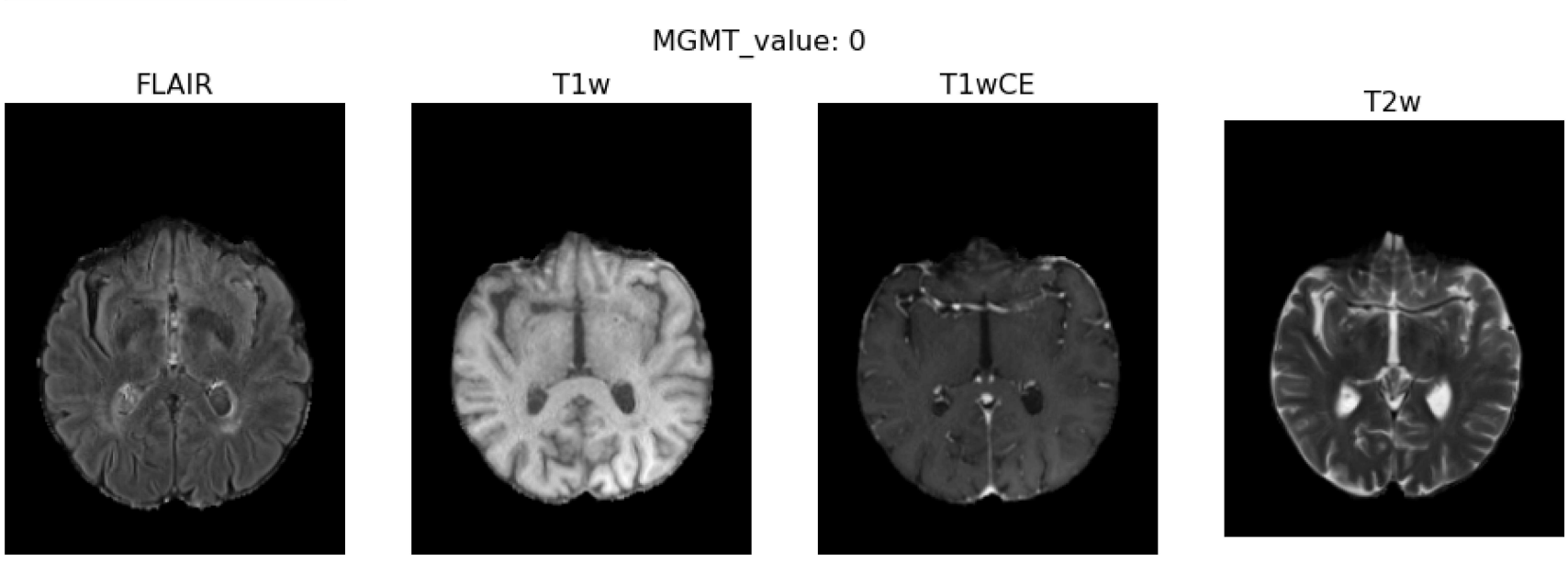

In [4]:
```
def load_dicom(path):
    dicom = pydicom.read_file(path)
    data = dicom.pixel_array
    data = data - np.min(data)
    if np.max(data) != 0:
        data = data / np.max(data)
    data = (data * 255).astype(np.uint8)
    return data
```

```
def visualize_sample(
    brats21id,
    slice_i,
    mgmt_value,
    types=("FLAIR", "T1w", "T1wCE", "T2w")
):
    plt.figure(figsize=(16, 5))
    patient_path = os.path.join(
        "../input/rsna-miccai-brain-tumor-radiogenomic-classification/train/",
        str(brats21id).zfill(5),
    )
    for i, t in enumerate(types, 1):
        t_paths = sorted(
            glob.glob(os.path.join(patient_path, t, "*")),
            key=lambda x: int(x[:-4].split("-")[-1]),
        )
        data = load_dicom(t_paths[int(len(t_paths) * slice_i)])
        plt.subplot(1, 4, i)
        plt.imshow(data, cmap="gray")
        plt.title(f"{t}", fontsize=16)
        plt.axis("off")

    plt.suptitle(f"MGMT_value: {mgmt_value}", fontsize=16)
    plt.show()
```

In [5]:
```
for i in random.sample(range(train_df.shape[0]), 10):
    _brats21id = train_df.iloc[i]["BraTS21ID"]
    _mgmt_value = train_df.iloc[i]["MGMT_value"]
    visualize_sample(brats21id=_brats21id, mgmt_value=_mgmt_value, slice_i=0.5)
```

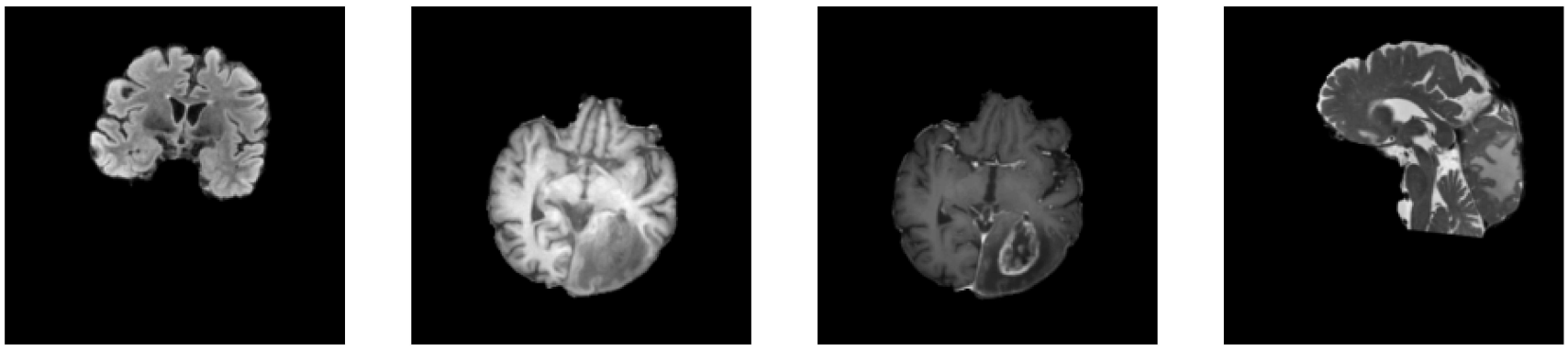


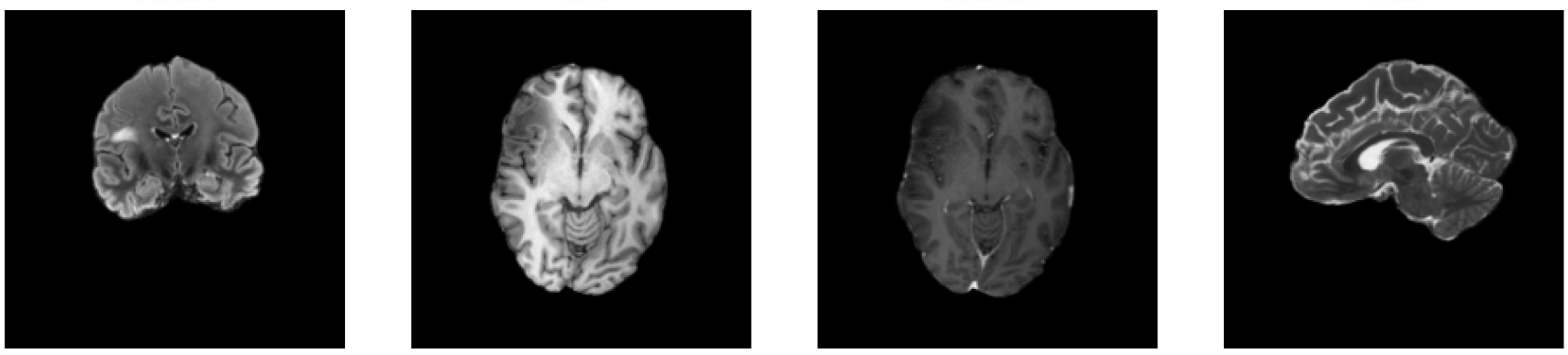
MGMT_value: 1

FLAIR T1w T1wCE T2w



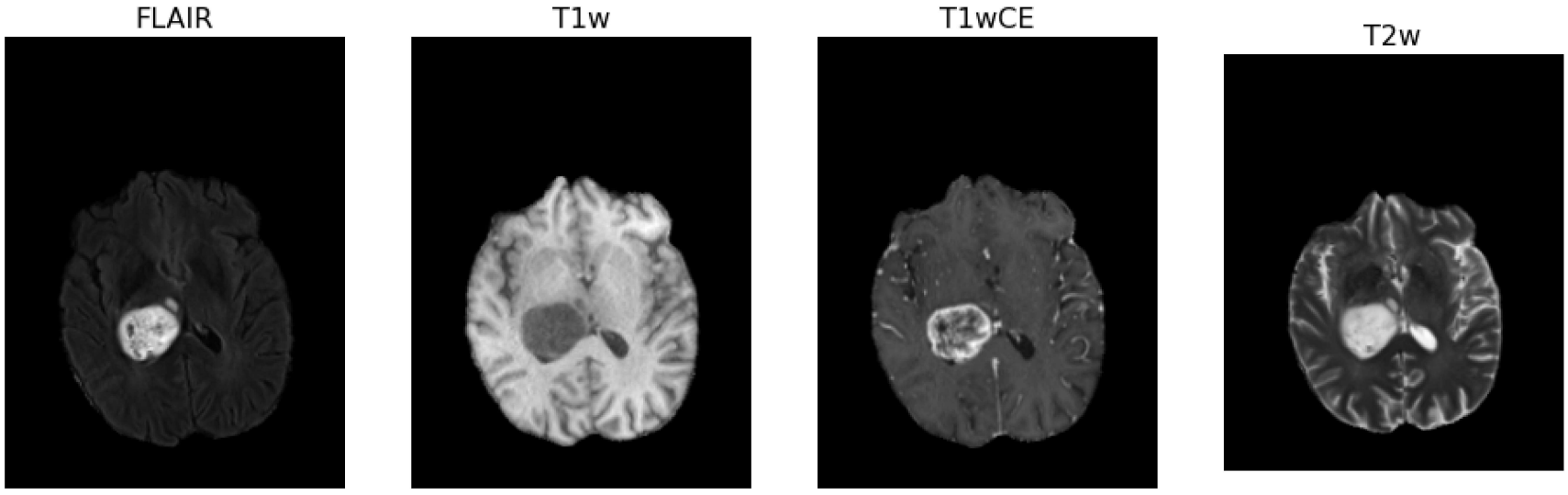
MGMT_value: 1

FLAIR T1w T1wCE T2w



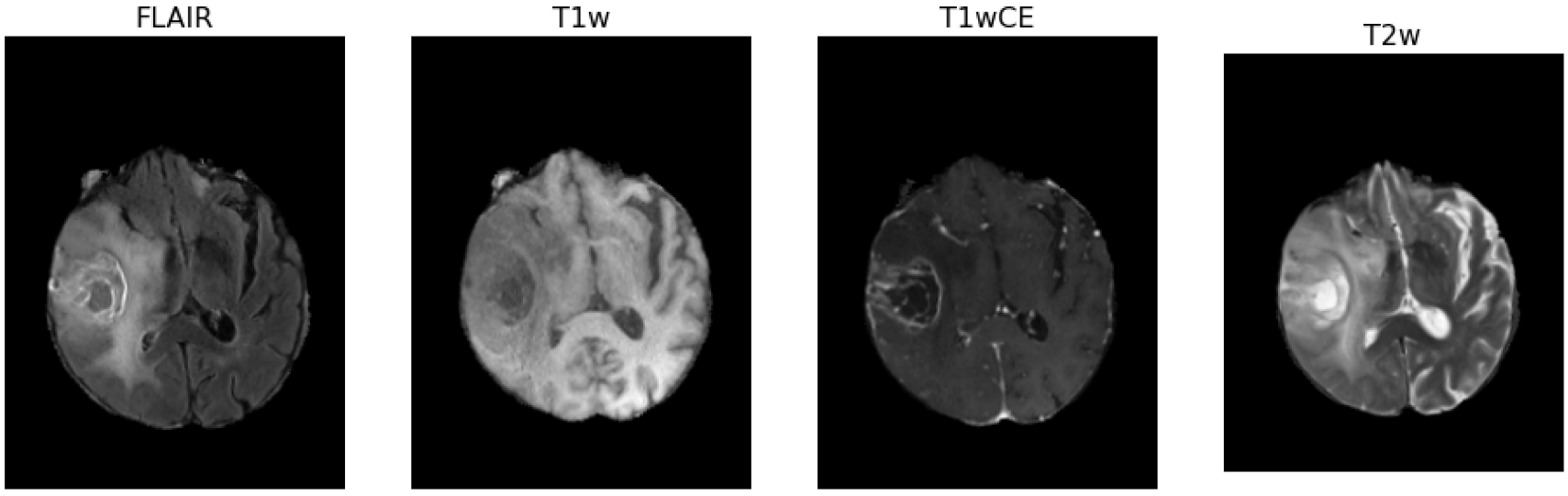
MGMT_value: 1

FLAIR T1w T1wCE T2w



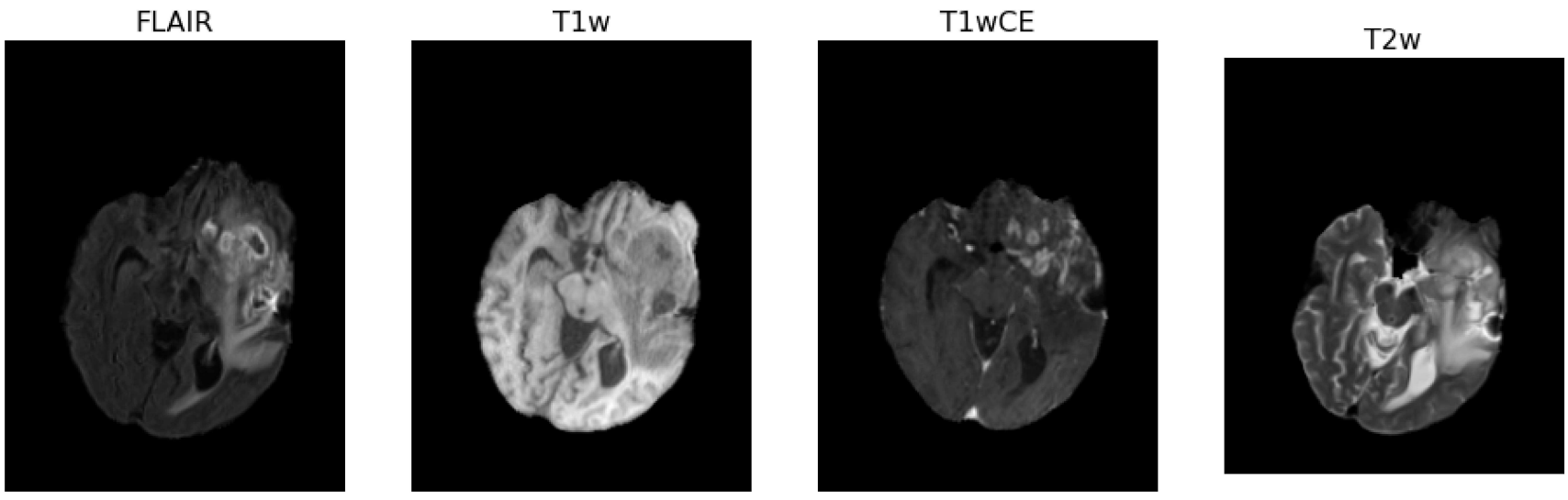
MGMT_value: 0

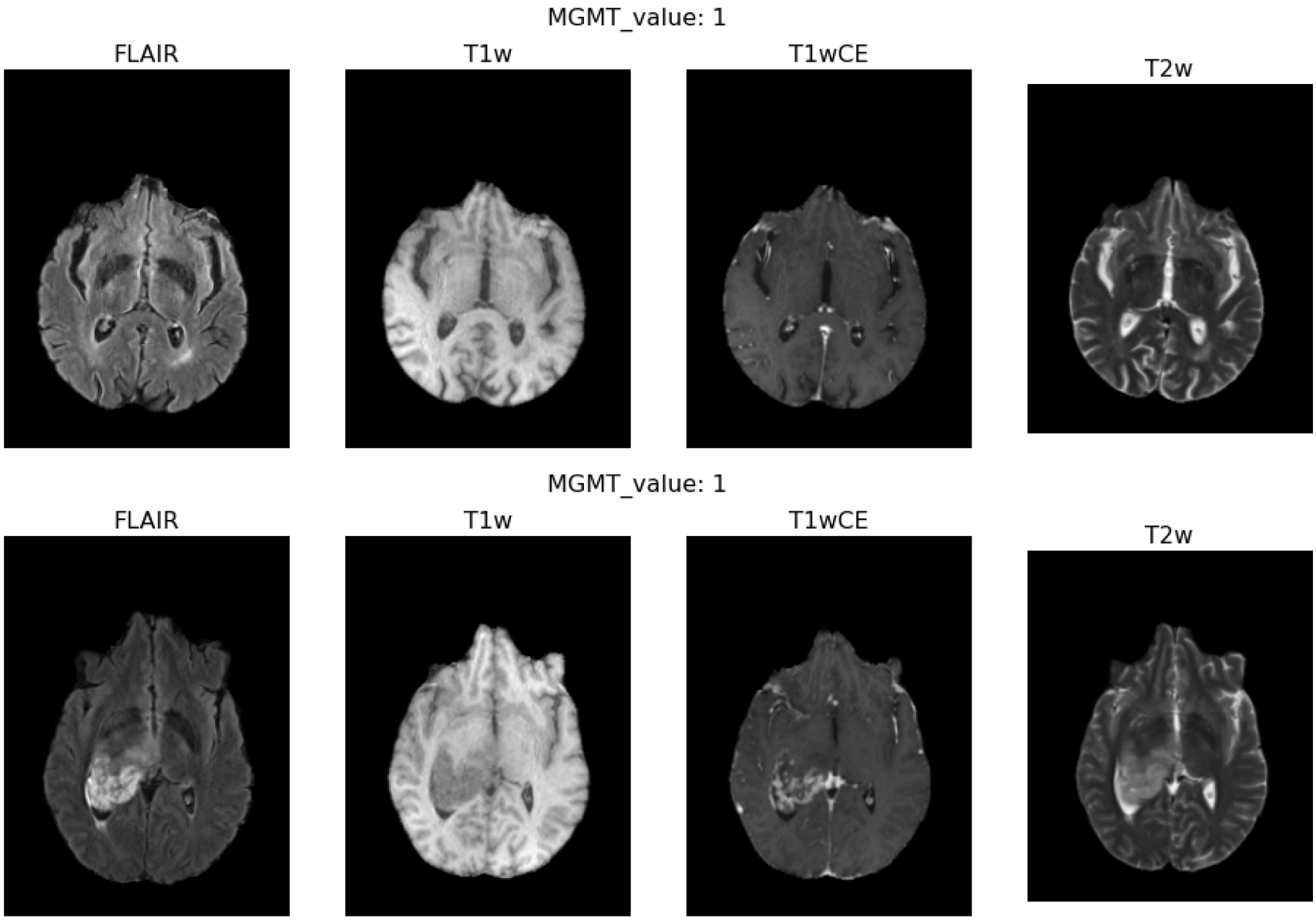
FLAIR T1w T1wCE T2w



MGMT_value: 0

FLAIR T1w T1wCE T2w



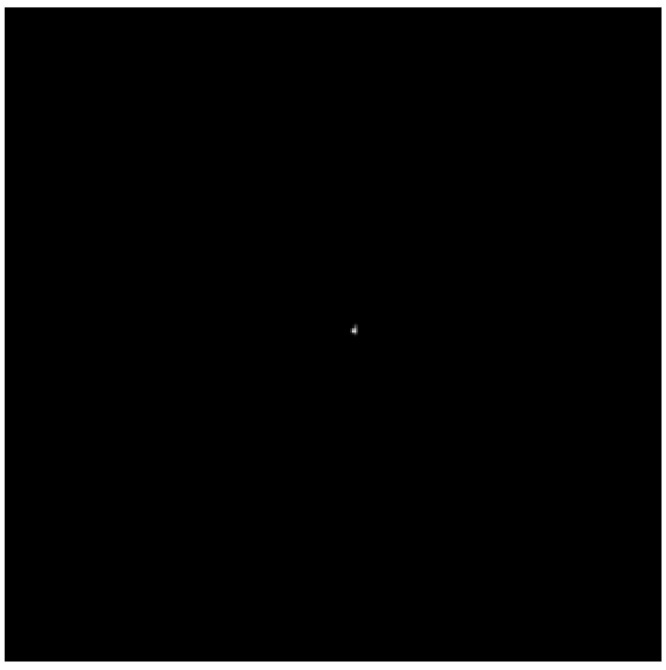


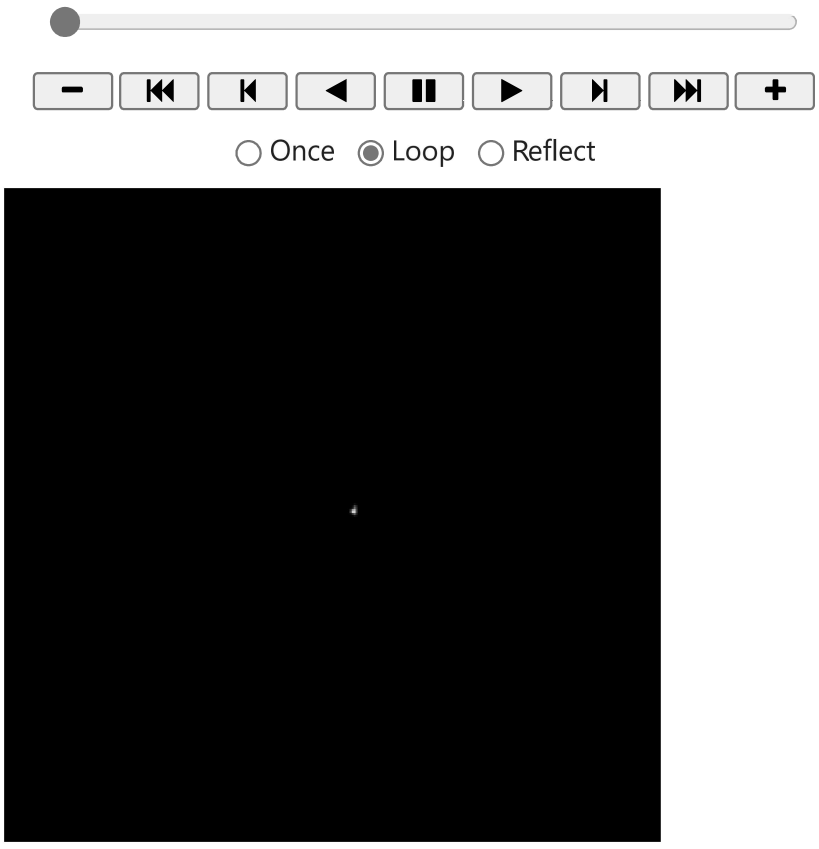

In [6]:
```
from matplotlib import animation, rc
rc('animation', html='jshtml')

def create_animation(ims):
    fig = plt.figure(figsize=(6, 6))
    plt.axis('off')
    im = plt.imshow(ims[0], cmap="gray")

    def animate_func(i):
        im.set_array(ims[i])
        return [im]

    return animation.FuncAnimation(fig, animate_func, frames = len(ims), interval = 1000//24)
```

In [7]:
```
def load_dicom_line(path):
    t_paths = sorted(
        glob.glob(os.path.join(path, "*")),
        key=lambda x: int(x[:-4].split("-")[-1]),
    )
    images = []
    for filename in t_paths:
        data = load_dicom(filename)
        if data.max() == 0:
            continue
        images.append(data)

    return images
```

In [8]:
```
images = load_dicom_line("../input/rsna-miccai-brain-tumor-radiogenomic-classification/train/00000/FLAIR")
create_animation(images)
```

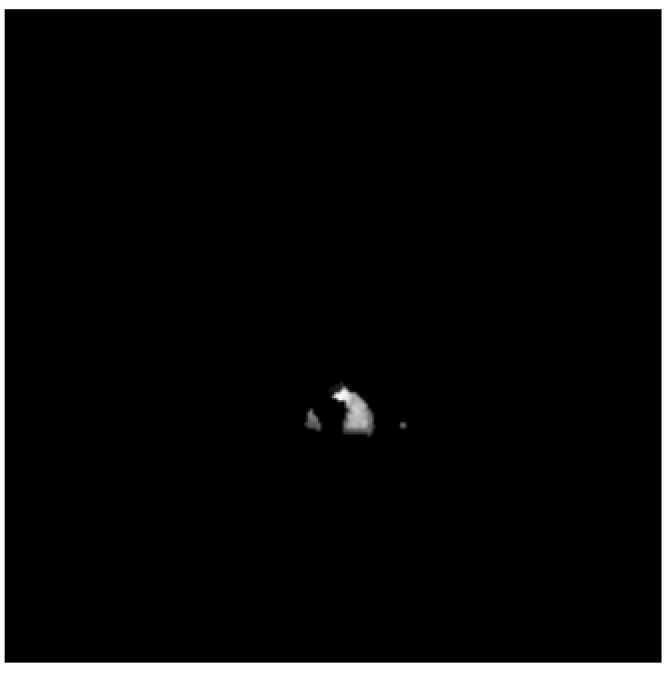
Out[8]:

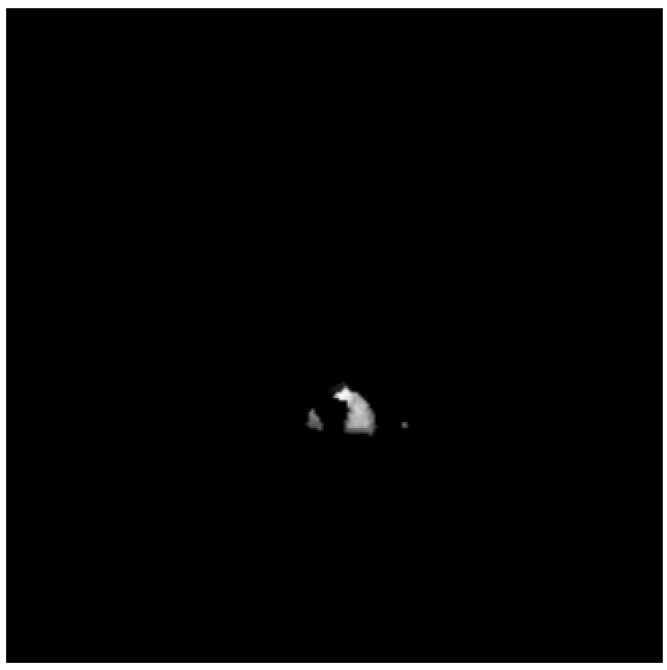




In [9]:
```
images = load_dicom_line("../input/rsna-miccai-brain-tumor-radiogenomic-classification/train/00000/T1w")
create_animation(images)
```

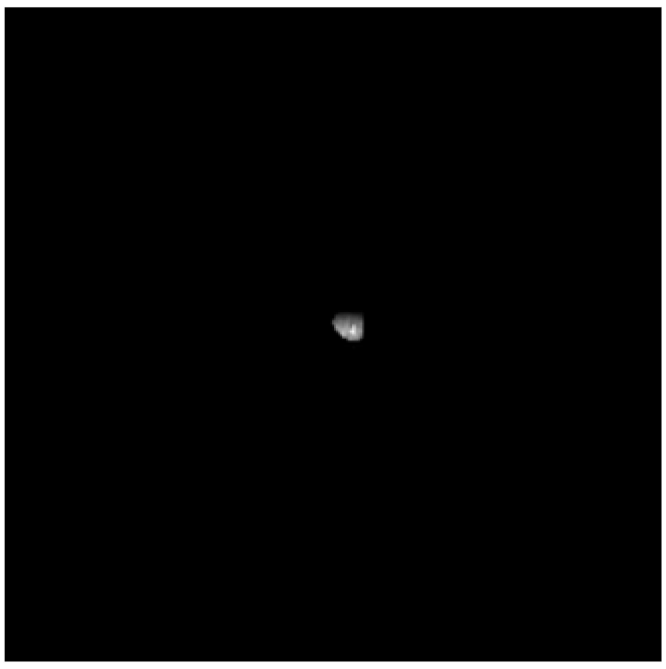
Out[9]:



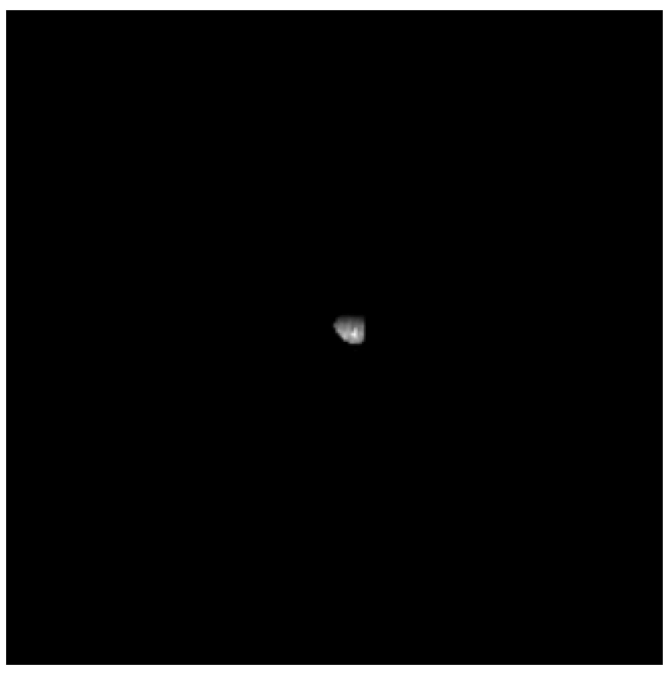


In [10]:
```
images = load_dicom_line("../input/rsna-miccai-brain-tumor-radiogenomic-classification/train/00000/T1wCE")
create_animation(images)
```

Out[10]:

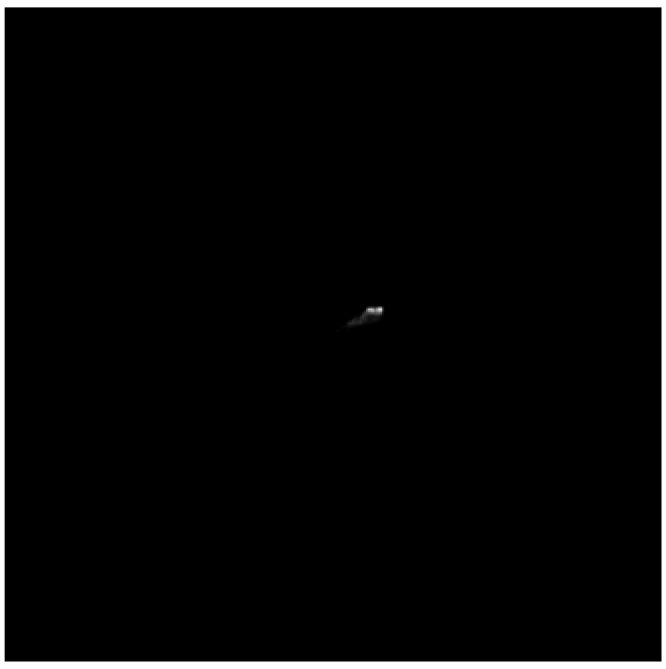


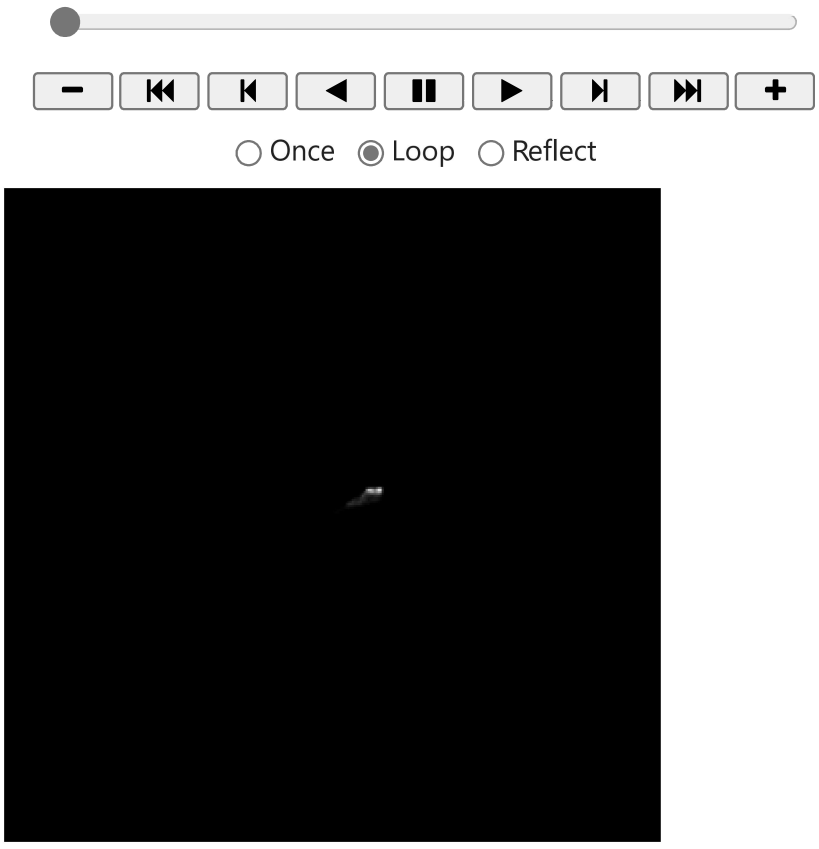
Once Loop Reflect



In [11]:
```
images = load_dicom_line("../input/rsna-miccai-brain-tumor-radiogenomic-classification/train/00000/T2w")
create_animation(images)
```

Out[11]:

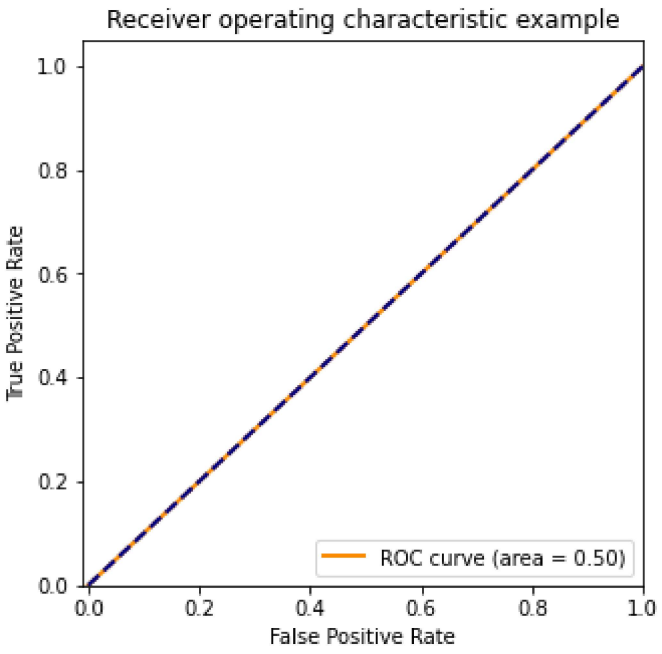


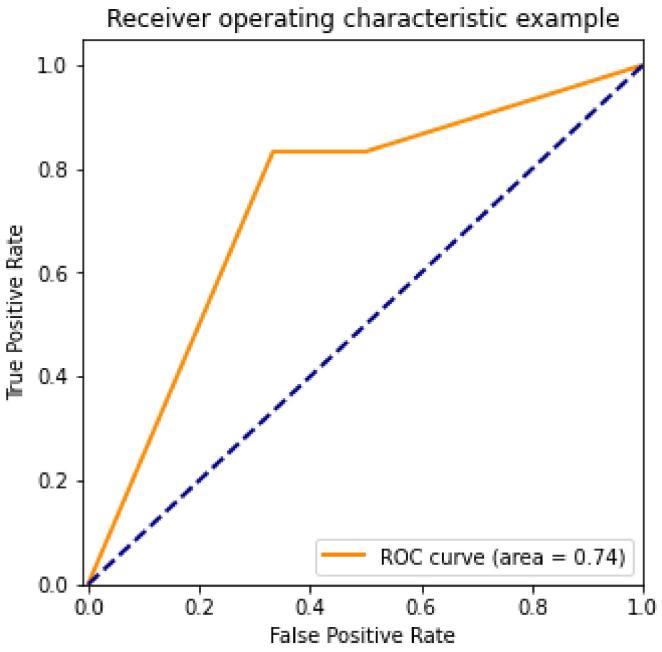


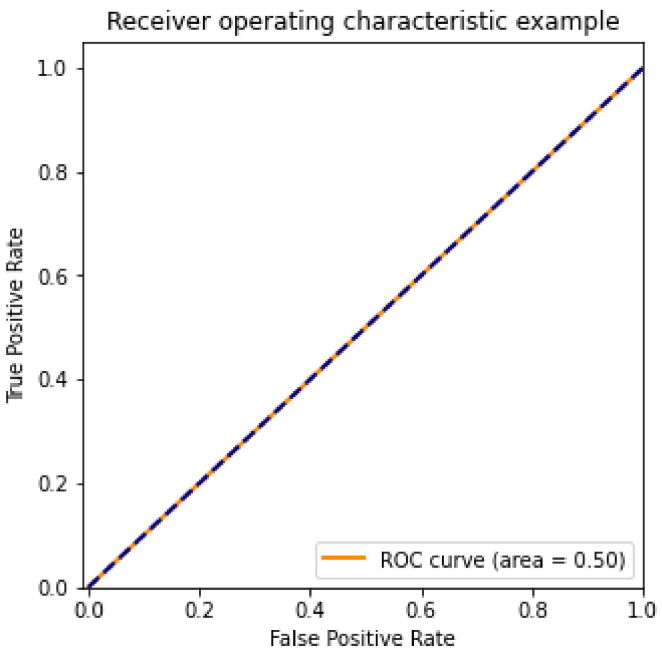
## Evaluation Metrics

Evaluated on [area under the ROC curve]between the predicted probability and the observed target.

In [12]:
```
from sklearn.metrics import roc_auc_score, roc_curve, auc

list_y_true = [
    [1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0.],
    [1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0.],
    [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.], #  IMBALANCE
]
list_y_pred = [
    [0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
    [0.9, 0.9, 0.9, 0.9, 0.1, 0.9, 0.9, 0.1, 0.9, 0.1, 0.1, 0.5],
    [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], #  IMBALANCE
]

for y_true, y_pred in zip(list_y_true, list_y_pred):
    fpr, tpr, _ = roc_curve(y_true, y_pred)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(5, 5))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([-0.01, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```




In [13]:
```
Result = pd.read_csv("../input/rsna-miccai-brain-tumor-radiogenomic-classification/sample_Result.csv")
# Result.to_csv("Result.csv", index=False)
Result
```

Out[13]:

| | BraTS21ID | MGMT_value |
|---|---|---|
| **0** | 1 | 0.5 |
| **1** | 13 | 0.5 |
| **2** | 15 | 0.5 |
| **3** | 27 | 0.5 |
| **4** | 37 | 0.5 |
| **...** | ... | ... |
| **82** | 826 | 0.5 |
| **83** | 829 | 0.5 |
| **84** | 833 | 0.5 |
| **85** | 997 | 0.5 |
| **86** | 1006 | 0.5 |

87 rows × 2 columns

## Modeling

In [14]:
```
package_path = "../input/efficientnet-pytorch/EfficientNet-PyTorch/EfficientNet-PyTorch-master/"
import sys
sys.path.append(package_path)

import time

import torch
from torch import nn
from torch.utils import data as torch_data
from sklearn import model_selection as sk_model_selection
from torch.nn import functional as torch_functional
import efficientnet_pytorch

from sklearn.model_selection import StratifiedKFold
```

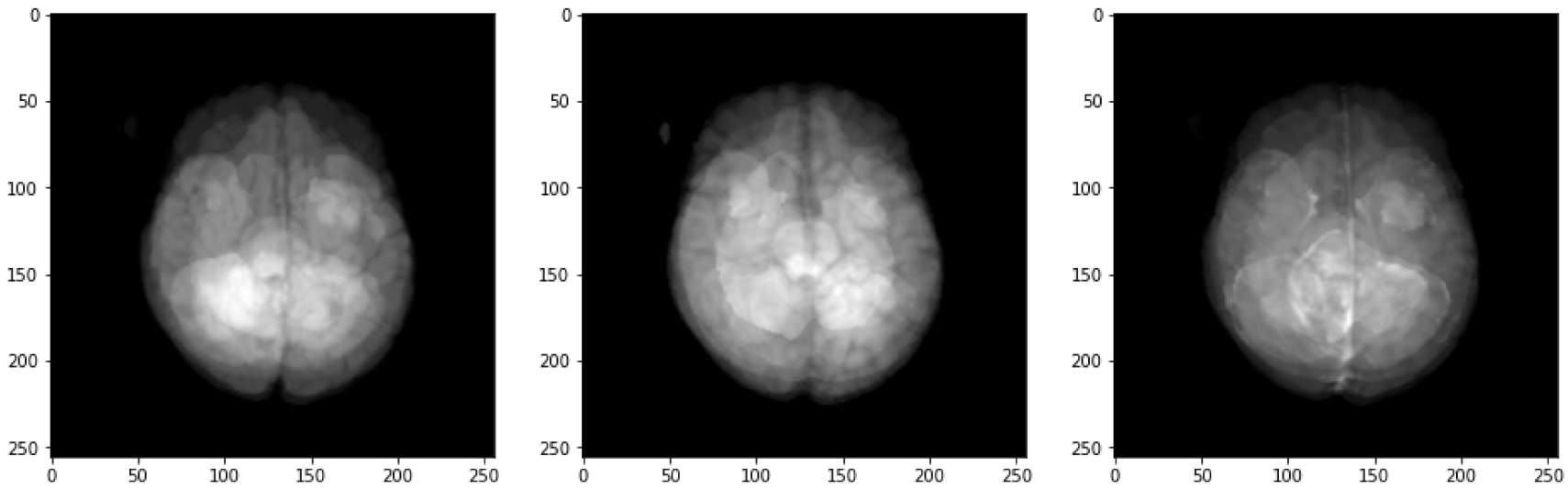
In [15]:
```
def set_seed(seed):
    random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True

set_seed(42)
```

In [16]:
```
df = pd.read_csv("../input/rsna-miccai-brain-tumor-radiogenomic-classification/train_labels.csv")
df_train, df_valid = sk_model_selection.train_test_split(
    df,
    test_size=0.2,
    random_state=42,
    stratify=train_df["MGMT_value"],
)
```

In [17]:
```
class DataRetriever(torch_data.Dataset):
    def __init__(self, paths, targets):
        self.paths = paths
        self.targets = targets

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, index):
        _id = self.paths[index]
        patient_path = f"../input/rsna-miccai-brain-tumor-radiogenomic-classification/train/{str(_id).zfill(5)}/"
        channels = []
        for t in ("FLAIR", "T1w", "T1wCE"): # "T2w"
            t_paths = sorted(
                glob.glob(os.path.join(patient_path, t, "*")),
                key=lambda x: int(x[:-4].split("-")[-1]),
            )
            # start, end = int(len(t_paths) * 0.475), int(len(t_paths) * 0.525)
            x = len(t_paths)
            if x < 10:
                r = range(x)
            else:
                d = x // 10
                r = range(d, x - d, d)

            channel = []
            # for i in range(start, end + 1):
            for i in r:
                channel.append(cv2.resize(load_dicom(t_paths[i]), (256, 256)) / 255)
            channel = np.mean(channel, axis=0)
            channels.append(channel)

        y = torch.tensor(self.targets[index], dtype=torch.float)

        return {"X": torch.tensor(channels).float(), "y": y}
```

In [18]:
```
train_data_retriever = DataRetriever(
    df_train["BraTS21ID"].values,
    df_train["MGMT_value"].values,
)

valid_data_retriever = DataRetriever(
    df_valid["BraTS21ID"].values,
    df_valid["MGMT_value"].values,
)
```

In [19]:
```
plt.figure(figsize=(16, 6))
for i in range(3):
    plt.subplot(1, 3, i + 1)
    plt.imshow(train_data_retriever[100]["X"].numpy()[i], cmap="gray")
```



In [20]:
```python
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = efficientnet_pytorch.EfficientNet.from_name("efficientnet-b0")
        checkpoint = torch.load("../input/efficientnet-pytorch/efficientnet-b0-08094119.pth")
        self.net.load_state_dict(checkpoint)
        n_features = self.net._fc.in_features
        self.net._fc = nn.Linear(in_features=n_features, out_features=1, bias=True)

    def forward(self, x):
        out = self.net(x)
        return out
```

In [21]:
```python
class LossMeter:
    def __init__(self):
        self.avg = 0
        self.n = 0

    def update(self, val):
        self.n += 1
        # incremental update
        self.avg = val / self.n + (self.n - 1) / self.n * self.avg


class AccMeter:
    def __init__(self):
        self.avg = 0
        self.n = 0

    def update(self, y_true, y_pred):
        y_true = y_true.cpu().numpy().astype(int)
        y_pred = y_pred.cpu().numpy() >= 0
        last_n = self.n
        self.n += len(y_true)
        true_count = np.sum(y_true == y_pred)
        # incremental update
        self.avg = true_count / self.n + last_n / self.n * self.avg
```

In [22]:
```python
class Trainer:
    def __init__(
        self,
        model,
        device,
        optimizer,
        criterion,
        loss_meter,
        score_meter
    ):
        self.model = model
        self.device = device
        self.optimizer = optimizer
        self.criterion = criterion
        self.loss_meter = loss_meter
        self.score_meter = score_meter

        self.best_valid_score = -np.inf
        self.n_patience = 0

        self.messages = {
            "epoch": "[Epoch {}: {}] loss: {:.5f}, score: {:.5f}, time: {} s",
            "checkpoint": "The score improved from {:.5f} to {:.5f}. Save model to '{}'",
            "patience": "\nValid score didn't improve last {} epochs."
        }

    def fit(self, epochs, train_loader, valid_loader, save_path, patience):
        for n_epoch in range(1, epochs + 1):
            self.info_message("EPOCH: {}", n_epoch)

            train_loss, train_score, train_time = self.train_epoch(train_loader)
            valid_loss, valid_score, valid_time = self.valid_epoch(valid_loader)

            self.info_message(
                self.messages["epoch"], "Train", n_epoch, train_loss, train_score, train_time
            )

            self.info_message(
                self.messages["epoch"], "Valid", n_epoch, valid_loss, valid_score, valid_time
```

```
            )

            if True:
#             if self.best_valid_score < valid_score:
                self.info_message(
                    self.messages["checkpoint"], self.best_valid_score, valid_score, save_path
                )
                self.best_valid_score = valid_score
                self.save_model(n_epoch, save_path)
                self.n_patience = 0
            else:
                self.n_patience += 1

            if self.n_patience >= patience:
                self.info_message(self.messages["patience"], patience)
                break

    def train_epoch(self, train_loader):
        self.model.train()
        t = time.time()
        train_loss = self.loss_meter()
        train_score = self.score_meter()

        for step, batch in enumerate(train_loader, 1):
            X = batch["X"].to(self.device)
            targets = batch["y"].to(self.device)
            self.optimizer.zero_grad()
            outputs = self.model(X).squeeze(1)

            loss = self.criterion(outputs, targets)
            loss.backward()

            train_loss.update(loss.detach().item())
            train_score.update(targets, outputs.detach())

            self.optimizer.step()

            _loss, _score = train_loss.avg, train_score.avg
            message = 'Train Step {}/{}, train_loss: {:.5f}, train_score: {:.5f}'
            self.info_message(message, step, len(train_loader), _loss, _score, end="\r")

        return train_loss.avg, train_score.avg, int(time.time() - t)

    def valid_epoch(self, valid_loader):
        self.model.eval()
        t = time.time()
        valid_loss = self.loss_meter()
        valid_score = self.score_meter()

        for step, batch in enumerate(valid_loader, 1):
            with torch.no_grad():
                X = batch["X"].to(self.device)
                targets = batch["y"].to(self.device)

                outputs = self.model(X).squeeze(1)
                loss = self.criterion(outputs, targets)

                valid_loss.update(loss.detach().item())
                valid_score.update(targets, outputs)

            _loss, _score = valid_loss.avg, valid_score.avg
            message = 'Valid Step {}/{}, valid_loss: {:.5f}, valid_score: {:.5f}'
            self.info_message(message, step, len(valid_loader), _loss, _score, end="\r")

        return valid_loss.avg, valid_score.avg, int(time.time() - t)

    def save_model(self, n_epoch, save_path):
        torch.save(
            {
                "model_state_dict": self.model.state_dict(),
                "optimizer_state_dict": self.optimizer.state_dict(),
                "best_valid_score": self.best_valid_score,
                "n_epoch": n_epoch,
            },
            save_path,
        )

    @staticmethod
    def info_message(message, *args, end="\n"):
        print(message.format(*args), end=end)
```

In [23]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

train_data_retriever = DataRetriever(
    df_train["BraTS21ID"].values,
    df_train["MGMT_value"].values,
)

valid_data_retriever = DataRetriever(
    df_valid["BraTS21ID"].values,
    df_valid["MGMT_value"].values,
)

train_loader = torch_data.DataLoader(
    train_data_retriever,
```

```
    batch_size=8,
    shuffle=True,
    num_workers=8,
)

valid_loader = torch_data.DataLoader(
    valid_data_retriever,
    batch_size=8,
    shuffle=False,
    num_workers=8,
)

model = Model()
model.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = torch_functional.binary_cross_entropy_with_logits

trainer = Trainer(
    model,
    device,
    optimizer,
    criterion,
    LossMeter,
    AccMeter
)

history = trainer.fit(
    10,
    train_loader,
    valid_loader,
    f"best-model-0.pth",
    100,
)
```

```
EPOCH: 1
[Epoch Train: 1] loss: 0.72701, score: 0.51496, time: 31 s7
[Epoch Valid: 1] loss: 0.72900, score: 0.52137, time: 7 s
The score improved from -inf to 0.52137. Save model to 'best-model-0.pth'
EPOCH: 2
[Epoch Train: 2] loss: 0.68856, score: 0.56197, time: 25 s7
[Epoch Valid: 2] loss: 0.70479, score: 0.52137, time: 5 s
The score improved from 0.52137 to 0.52137. Save model to 'best-model-0.pth'
EPOCH: 3
[Epoch Train: 3] loss: 0.69817, score: 0.54060, time: 25 s8
[Epoch Valid: 3] loss: 0.70880, score: 0.48718, time: 5 s
The score improved from 0.52137 to 0.48718. Save model to 'best-model-0.pth'
EPOCH: 4
[Epoch Train: 4] loss: 0.68301, score: 0.58333, time: 25 s2
[Epoch Valid: 4] loss: 0.68398, score: 0.51282, time: 5 s
The score improved from 0.48718 to 0.51282. Save model to 'best-model-0.pth'
EPOCH: 5
[Epoch Train: 5] loss: 0.67145, score: 0.58974, time: 25 s6
[Epoch Valid: 5] loss: 0.70245, score: 0.39316, time: 6 s
The score improved from 0.51282 to 0.39316. Save model to 'best-model-0.pth'
EPOCH: 6
[Epoch Train: 6] loss: 0.66009, score: 0.61111, time: 25 s2
[Epoch Valid: 6] loss: 0.71838, score: 0.51282, time: 6 s
The score improved from 0.39316 to 0.51282. Save model to 'best-model-0.pth'
EPOCH: 7
[Epoch Train: 7] loss: 0.60412, score: 0.67521, time: 25 s4
[Epoch Valid: 7] loss: 2.06478, score: 0.46154, time: 5 s
The score improved from 0.51282 to 0.46154. Save model to 'best-model-0.pth'
EPOCH: 8
[Epoch Train: 8] loss: 0.56895, score: 0.70726, time: 25 s3
[Epoch Valid: 8] loss: 1.03076, score: 0.62393, time: 5 s
The score improved from 0.46154 to 0.62393. Save model to 'best-model-0.pth'
EPOCH: 9
[Epoch Train: 9] loss: 0.51941, score: 0.76282, time: 25 s6
[Epoch Valid: 9] loss: 1.23480, score: 0.55556, time: 5 s
The score improved from 0.62393 to 0.55556. Save model to 'best-model-0.pth'
EPOCH: 10
[Epoch Train: 10] loss: 0.39818, score: 0.81410, time: 25 s
[Epoch Valid: 10] loss: 0.88553, score: 0.55556, time: 5 s
The score improved from 0.55556 to 0.55556. Save model to 'best-model-0.pth'
```

In [24]:
```
models = []
for i in range(1):
    model = Model()
    model.to(device)

    checkpoint = torch.load(f"best-model-{i}.pth")
    model.load_state_dict(checkpoint["model_state_dict"])
    model.eval()

    models.append(model)
```

In [25]:
```
class DataRetriever(torch_data.Dataset):
    def __init__(self, paths):
        self.paths = paths

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, index):
```

```
        _id = self.paths[index]
        patient_path = f"../input/rsna-miccai-brain-tumor-radiogenomic-classification/test/{str(_id).zfill(5)}/"
        channels = []
        for t in ("FLAIR", "T1w", "T1wCE"): # "T2w"
            t_paths = sorted(
                glob.glob(os.path.join(patient_path, t, "*")),
                key=lambda x: int(x[:-4].split("-")[-1]),
            )
            # start, end = int(len(t_paths) * 0.475), int(len(t_paths) * 0.525)
            x = len(t_paths)
            if x < 10:
                r = range(x)
            else:
                d = x // 10
                r = range(d, x - d, d)

            channel = []
            # for i in range(start, end + 1):
            for i in r:
                channel.append(cv2.resize(load_dicom(t_paths[i]), (256, 256)) / 255)
            channel = np.mean(channel, axis=0)
            channels.append(channel)

        return {"X": torch.tensor(channels).float(), "id": _id}
```

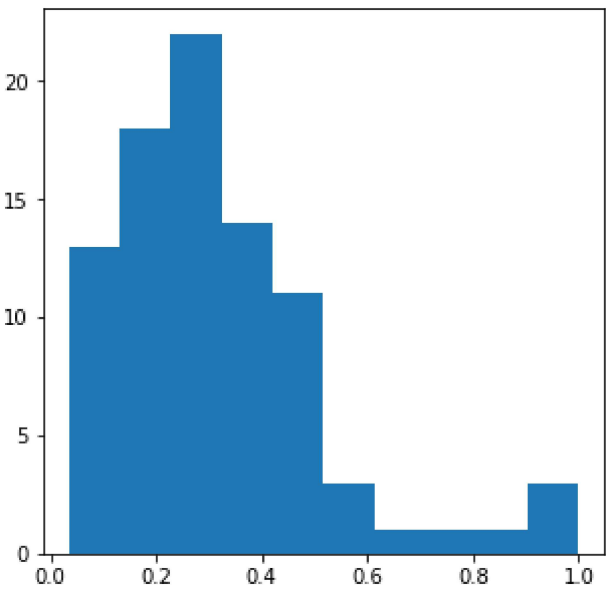
In [26]:
```
submission = pd.read_csv("../input/rsna-miccai-brain-tumor-radiogenomic-classification/sample_submission.csv")

test_data_retriever = DataRetriever(
    submission["BraTS21ID"].values,
)

test_loader = torch_data.DataLoader(
    test_data_retriever,
    batch_size=4,
    shuffle=False,
    num_workers=8,
)
```

In [27]:
```
y_pred = []
ids = []

for e, batch in enumerate(test_loader):
    print(f"{e}/{len(test_loader)}", end="\r")
    with torch.no_grad():
        tmp_pred = np.zeros((batch["X"].shape[0], ))
        for model in models:
            tmp_res = torch.sigmoid(model(batch["X"].to(device))).cpu().numpy().squeeze()
            tmp_pred += tmp_res
        y_pred.extend(tmp_pred)
        ids.extend(batch["id"].numpy().tolist())
```

```
21/22
```

In [28]:
```
Result= pd.DataFrame({"BraTS21ID": ids, "MGMT_value": y_pred})
Result.to_csv("Result.csv", index=False)
```

In [29]:
```
plt.figure(figsize=(5, 5))
plt.hist(Result["MGMT_value"]);
```




In [30]:
```
Result
```

Out[30]:

| | BraTS21ID | MGMT_value |
|---|---|---|
| **0** | 1 | 0.176131 |
| **1** | 13 | 0.753934 |
| **2** | 15 | 0.506579 |
| **3** | 27 | 0.407434 |
| **4** | 37 | 0.351310 |
| **...** | ... | ... |
| **82** | 826 | 0.476697 |
| **83** | 829 | 0.373890 |
| **84** | 833 | 0.232642 |
| **85** | 997 | 0.645033 |
| **86** | 1006 | 0.087923 |

87 rows × 2 columns