summarize

May 16, 2023

1

matplotlib

1.1 Loss

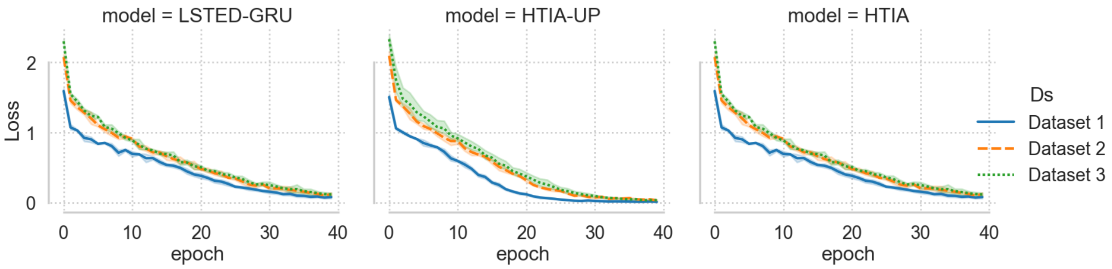
[2]:

```
# loss  GRu
import os
import pandas as pd
import matplotlib.pyplot as plt

#  excel

dir_path1= r'lstm_attn\    \sad'
dir_path2= r'  \lstm_attn\    \sad'
dir_path3= r'  \ gru\    \sad'  # USED-up   Loss3
#
def input_data(dir_path):
    file_names = os.listdir(dir_path)

    #     loss
    all_losses = {}

    #
    count = 0
    group_dict = {}

    #    excel     loss     epoch
    for file_name in file_names:
        file_path = os.path.join(dir_path, file_name)
        df = pd.read_excel(file_path)

        #
        if count % 5 == 0:
            group_name = 'Dataset {}'.format(int(count/5)+1)
            group_dict[group_name] = []

        #   epoch  loss
        model_name = file_name.split(".")[0]
        epoch_losses = []
        for epoch in range(0, 100):
            batch_losses = df[df['Step'] == epoch]['Value'].tolist()
            epoch_loss = sum(batch_losses) / len(batch_losses)
            epoch_losses.append(epoch_loss)

        #  loss
        group_dict[group_name].append(epoch_losses)

        count += 1

    #            loss
    for group_name, group_losses in group_dict.items():
        epoch_losses_avg = []
        for epoch in range(0, 100):
            epoch_losses_sum = 0
            for model_losses in group_losses:
                epoch_losses_sum += model_losses[epoch]
            epoch_losses_avg.append(epoch_losses_sum/len(group_losses))
        all_losses[group_name] = epoch_losses_avg
#     fig, ax = plt.subplots(figsize=(8, 6), dpi=300)
    colors = ['r', 'g', 'b']
    data_list = []
    label_list = []
    for i, (group_name, group_losses) in enumerate(group_dict.items()):
        for j, model_losses in enumerate(group_losses):
#             ax.plot(model_losses[0:40], color=colors[i], label='{}({})'.
 ↪format(group_name,j+1))
            data_list.append(model_losses[0:40])
            label_list.append('{}'.format(group_name))
    #         label_list.append('{}({})'.format(group_name, j+1))
    df = pd.DataFrame(data_list).T
    df.columns = label_list
    df['epoch']=df.index
    df=df.melt(id_vars=['epoch'],
        value_vars=['Dataset 1', 'Dataset 1', 'Dataset 1', 'Dataset 1',␣
 ↪'Dataset 1',
       'Dataset 2', 'Dataset 2', 'Dataset 2', 'Dataset 2', 'Dataset 2',
       'Dataset 3', 'Dataset 3', 'Dataset 3', 'Dataset 3', 'Dataset 3'],
        var_name=["Ds"],
        value_name="Loss"
       )
    return df
df1=input_data(dir_path1)
df2=input_data(dir_path2)
df3=input_data(dir_path3)
```

```
data=pd.concat([df2,df3,df1],keys=['LSTED-GRU','HTIA-UP','HTIA'])
data['model']=data.index.get_level_values(0)
```

```
[3]: import seaborn as sns

sns.set_style("whitegrid", {
    "grid.linestyle": ":",
    "lines.solid_capstyle":'round',
})
sns.set_context('poster')
# Plot the responses for different events and regions
plt.figure(figsize=(15, 10))
g=sns.relplot(x="epoch", y="Loss",
             hue="Ds", style="Ds",kind='line',
#              errorbar="sd",
            col='model',
             data=data)
g.add_legend(frameon=True)
g.legend.set_bbox_to_anchor((1.1, 0.5))
sns.despine(offset=2, trim=True)
plt.savefig("pdf\Loss_plot.pdf",dpi=600)
```

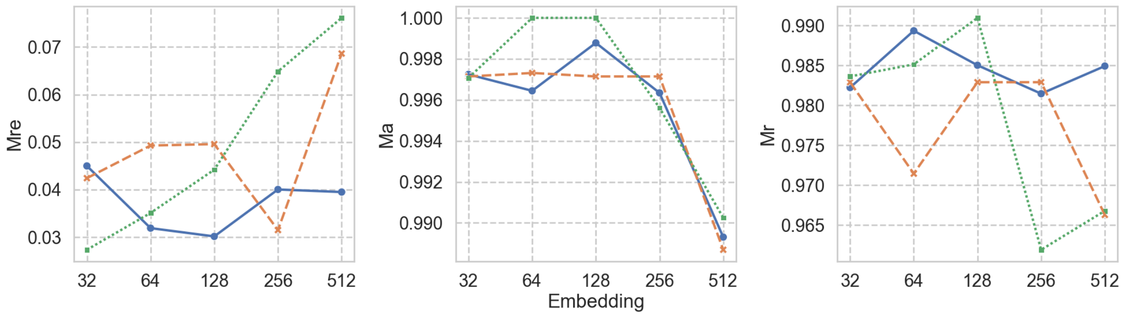
```
<Figure size 1500x1000 with 0 Axes>
```



1.2

```
[3]: import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="ticks")

import pandas as pd
data=pd.read_csv("    \embedding.csv")
data['embedding']=pd.Series(list((32,64,128,256,512)*3)).astype(str)
data['data']=data['data'].astype(str)
data.columns=['data', 'manu_seed', 'hidden_size', 'bsize', 'lr', 'Mre', 'Ma',␣
 ↪'Mr',
       'Mre1', 'Ma1', 'embedding']
```

```
sns.set_style("whitegrid")


plt.figure(figsize=(20,6))
sns.set_context('poster')
pc = sns.color_palette('deep', 3)
plt.rcParams["grid.linestyle"] = "--"
plt.rcParams['axes.linewidth'] = 2
plt.subplot(1,3,1)   #221
# sns.set_context('poster')
sns.lineplot(x='embedding', y='Mre',palette=pc ,
             hue='data',data=data, legend=False, style='data')

sns.scatterplot(x='embedding', y='Mre',palette=pc, s=100,
                hue='data', data=data, legend=False, style='data')
plt.xlabel("")
plt.subplot(1,3,2)
sns.lineplot(x='embedding', y='Ma',palette=pc ,
             hue='data',data=data, legend=False, style='data')
sns.scatterplot(x='embedding', y='Ma',palette=pc, s=100,
                hue='data', data=data, legend=False, style='data')

plt.xlabel("Embedding")



plt.subplot(1,3,3)

sns.lineplot(x='embedding', y='Mr',palette=pc ,
             hue='data',data=data, legend=False, style='data')
sns.scatterplot(x='embedding', y='Mr',palette=pc, s=100,
                hue='data', data=data, legend=False, style='data')
# plt.legend(loc='center left', bbox_to_anchor=(1.05, 0.5))

plt.xlabel("")
plt.tight_layout()  #
plt.savefig ( "pdf/embedding.pdf", dpi= 600)
```

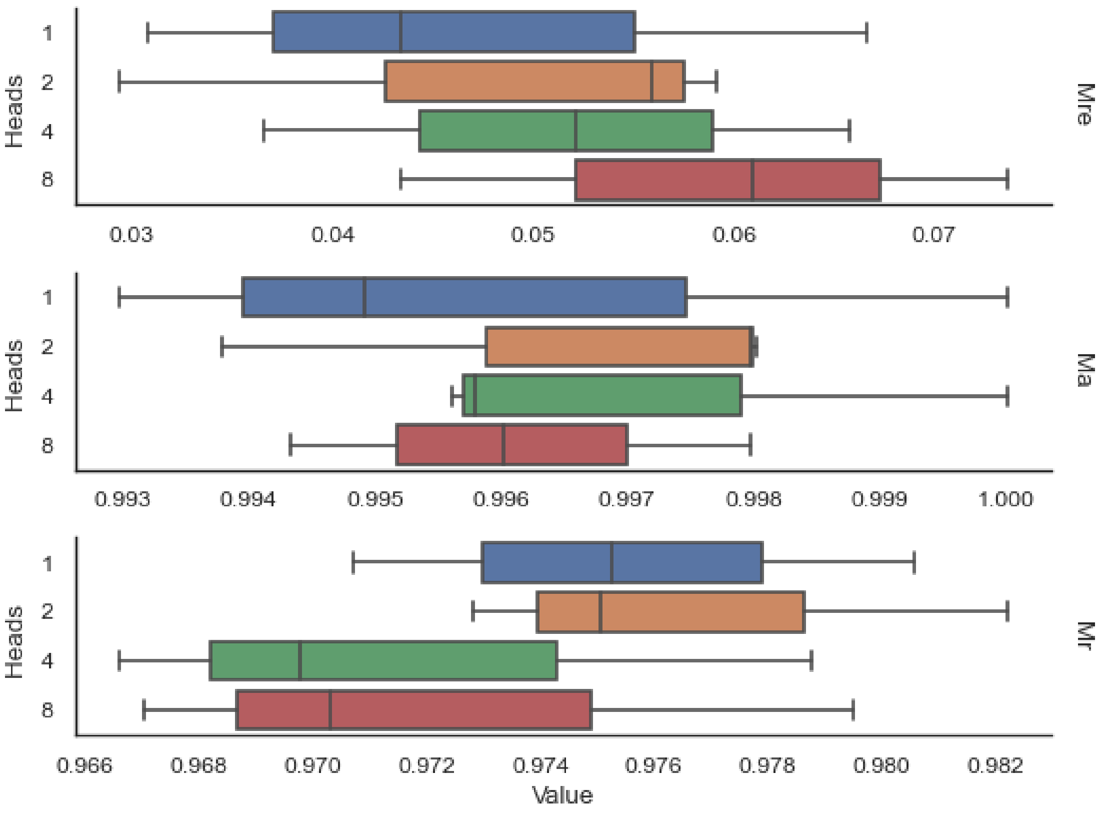


1.3

Z Z

[34]:

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("    \head.csv")
data['head']=['1','2','4','8']*3
data=data[['data','head', 'Mre3', 'Ma3', 'Mr3',]]
df=data.melt(id_vars=['data','head'],
        value_vars=['Mre3', 'Ma3', 'Mr3',],
        var_name="Index",
        value_name="Value"
       )
df['Index']=df['Index'].str.split("3",expand=True)[0]
# plt.figure(figsize=(20,12))
sns.set_context('paper')
g = sns.catplot(
    data=df,
    x="Value", y="head", row="Index",legend='strip',legend_out=True,
    kind="box", orient="h",
    sharex=False, margin_titles=True,
    height=1.5, aspect=4,
)
g.set(ylabel="Heads  ")

g.set_titles(row_template="{row_name} ")
plt.savefig('pdf\plotheading.pdf', dpi=300)
```



1.4 chord

d3block

[145]:
```
import seaborn as sns
import matplotlib.pyplot as plt

import pickle
import matplotlib.colors as mcolors
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

import numpy as np
import pandas as pd
import torch
with open("   \\attn\\ numpairs  attr\\attn_2.pickle", "rb") as file:
   attn2= pickle.load( file).cpu()
with open("   \\attn\\ numpairs  attr\\env2.pickle", "rb") as file:
   environment2= pickle.load( file)
data2=attn2[:,0,:,:].squeeze(0).numpy() #
```

```
numpairs,attr,label=environment2
numpair_numpy=numpairs.squeeze(0).cpu().numpy().astype(str)
dataframe2=pd.DataFrame(data2,index=numpair_numpy,columns=numpair_numpy)
new_df = dataframe2.drop(index='0',columns='0').stack().reset_index()
new_df.columns = ['source', 'target', 'weight']
new_df
```

[145]:

```
     source target        weight
0         9      9  0.000000e+00
1         9      5  5.000000e-01
2         9      5  5.000000e-01
3         9      6  2.207841e-35
4         9      6  2.207841e-35
…       …    …             …
1439      4      4  0.000000e+00
1440      4      4  0.000000e+00
1441      4      6  0.000000e+00
1442      4      6  0.000000e+00
1443      4      4  0.000000e+00

[1444 rows x 3 columns]
```

2 ATTN

attn

[32]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import torch


def open_attn(data,manu_seed,index):
    with open(rf"lstm_attn\attn\attn_{index}_{data}_{manu_seed}.pickle","rb")␣
 ↪as file:
        attn= pickle.load( file).cpu()
    with open(rf"lstm_attn\attn\env_{index}_{data}_{manu_seed}.pickle", "rb")␣
 ↪as file:
        environment= pickle.load( file)
    with open(rf"lstm_attn\attn\pred_{index}_{data}_{manu_seed}.pickle", "rb")␣
 ↪as file:
         pred= pickle.load(file)
    data=attn[:,0,:,:].squeeze(0).numpy() #
    numpairs,attr,label=environment
    numpair_numpy=numpairs.squeeze(0).numpy()#.astype(str)
    dataframe=pd.DataFrame(data,index=numpair_numpy,columns=numpair_numpy)

    return attn,numpairs,attr,label,pred


def group_tensors_by_value(tensor_list):
    """
       tensor                 tensor
    :param tensor_list:     tensor
    :return:                tensor
    """
    #       tensor          tensor
    tensor_dict = {}
    for i, tensor in enumerate(tensor_list):
        found = False
        for key in tensor_dict:
            if torch.equal(key, tensor):
                tensor_dict[key].append(i)
                found = True
                break
        if not found:
            tensor_dict[tensor] = [i]

    #              tensor
    grouped_tensors = []
    index_list = []
    for tensor, indices in tensor_dict.items():
        if len(indices) > 1:
            grouped_tensor = torch.stack([tensor_list[i] for i in indices])
            grouped_tensors.append(grouped_tensor)
            index_list.append(indices)

    #
    return grouped_tensors, index_list

def attn_clean(attn,numpairs):
    numpair_numpy=numpairs.squeeze(0).cpu().numpy().astype(str)
    dataframe=pd.DataFrame(attn.squeeze(0)[0,:,:
 ↪],index=numpair_numpy,columns=numpair_numpy).drop(index='0',columns='0')
    return dataframe

def get_data(Ds,select_datas,length):
    manu_seeds = [f"{i}" for i in range(5)]
    txtpaths = ["981762","981808","981814"]
    numpair_list=[open_attn(Ds,i,j)[1] for i in manu_seeds for j in␣
 ↪range(length)]
    raw_list=[open_attn(Ds,i,j) for i in manu_seeds for j in range(length)]
    _,index=group_tensors_by_value(numpair_list)
    pred=[raw_list[i][-1] for i in select_datas]
    attr=[raw_list[i][-3] for i in select_datas]
    print(pred,attr)
    datas=[attn_clean(*raw_list[i][0:2]) for i in select_datas]
    return datas

ds1=get_data('981762',[ 163,136,106,68],164)
ds2=get_data('981808',[20, 65, 242, 246],70)
ds3=get_data('981814',[ 550, 492,152, 182 ],114)
datas=ds1+ds2+ds3
```

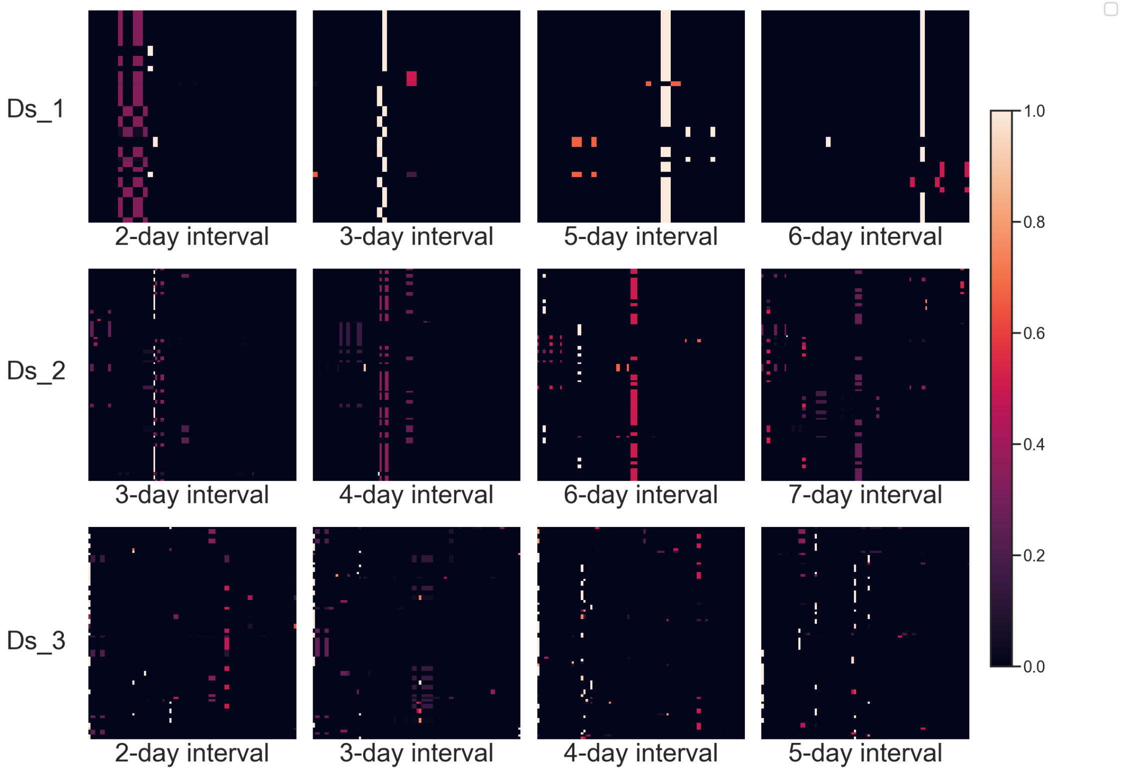
```
[[0.7, 1.0, 0.6], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]]
[tensor([[[3, 8],
         [4, 7],
         [5, 6],
         [6, 5],
         [0, 4],
         [1, 3],
         [2, 2]]]), tensor([[[3, 9],
         [4, 8],
         [5, 7],
         [6, 6],
         [0, 5],
         [1, 4],
         [2, 3]]]), tensor([[[ 3, 11],
         [ 4, 10],
         [ 5,  9],
         [ 6,  8],
         [ 0,  7],
         [ 1,  6],
         [ 2,  5]]]), tensor([[[ 3, 12],
         [ 4, 11],
         [ 5, 10],
         [ 6,  9],
         [ 0,  8],
         [ 1,  7],
         [ 2,  6]]])]
[[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.09090909090909091, 1.0,
0.9090909090909091], [0.0, 1.0, 1.0]] [tensor([[[3, 9],
         [4, 8],
         [5, 7],
         [6, 6],
         [0, 5],
         [1, 4],
         [2, 3]]]), tensor([[[ 3, 10],
         [ 4,  9],
         [ 5,  8],
         [ 6,  7],
         [ 0,  6],
         [ 1,  5],
         [ 2,  4]]]), tensor([[[ 3, 11],
         [ 4, 10],
         [ 5,  9],
         [ 6,  8],
         [ 0,  7],
         [ 1,  6],
         [ 2,  5]]]), tensor([[[ 3, 12],
         [ 4, 11],
         [ 5, 10],
         [ 6,  9],
         [ 0,  8],
         [ 1,  7],
         [ 2,  6]]])]
[[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]]
[tensor([[[6, 7],
         [0, 6],
         [1, 5],
         [2, 4],
         [3, 3],
         [4, 2],
         [5, 1]]]), tensor([[[6, 8],
         [0, 7],
         [1, 6],
         [2, 5],
         [3, 4],
         [4, 3],
         [5, 2]]]), tensor([[[6, 9],
         [0, 8],
         [1, 7],
         [2, 6],
         [3, 5],
         [4, 4],
         [5, 3]]]), tensor([[[ 6, 11],
         [ 0, 10],
         [ 1,  9],
         [ 2,  8],
         [ 3,  7],
         [ 4,  6],
         [ 5,  5]]])]
```

[36]:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#
data = np.random.rand(10, 10)

#   2x5       5x5
fig, axes = plt.subplots(3, 4, figsize=(20, 15),)

#
axes = axes.flatten()

#
for i, ax in enumerate(axes):

    sns.set_theme(style="ticks")
#     sns.set_context("talk")
    sns.set_context("poster")
    sns.color_palette("rocket",as_cmap=True)# as_cmap=True
    sns.heatmap(datas[i], ax=ax, cbar=False, xticklabels=False, yticklabels=False)
#     ax.set_title(f"Plot ") #

day=[2,3,5,6,3,4,6,7,2,3,4,5]
#
for i,ax in enumerate(axes):
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.set_xlabel(f"{day[i]}-day interval",fontsize=35)
#     ax.set_ylabel("b")
fig.legend()
fig.subplots_adjust(top=0.9, wspace=0.3, hspace=0)
#
# fig.text(0.5, 1, "X-Axis Title", ha="center", fontsize=22)
# fig.text(-0.02, 0.5, "Y-Axis Title", va="center", rotation="vertical", fontsize=22)
fig.text(-0.06, 0.52, "Ds_2", va="center", fontsize=35)
fig.text(-0.06, 0.18, "Ds_3", va="center", fontsize=35)
fig.text(-0.06, 0.85, "Ds_1", va="center",  fontsize=35)


# fig.text(0.15, 0.99, "Incidence attention weights when predicting future results at different time intervals", va="center", fontsize=22)
#
plt.tight_layout()
fig.subplots_adjust(right=0.85)  #
cbar_ax = fig.add_axes([0.87, 0.15, 0.02, 0.7])  #
fig.colorbar(axes[0].collections[0], cax=cbar_ax)  #
#
plt.savefig('pdf\\attn1.pdf', dpi=900)
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



[146]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import torch


def open_attn(data,manu_seed,index):
    with open(rf"lstm_attn\attn\attn_{index}_{data}_{manu_seed}.pickle","rb")␣
  ↪as file:
        attn= pickle.load( file).cpu()
    with open(rf"lstm_attn\attn\env_{index}_{data}_{manu_seed}.pickle", "rb")␣
  ↪as file:
        environment= pickle.load( file)
    with open(rf"lstm_attn\attn\pred_{index}_{data}_{manu_seed}.pickle", "rb")␣
  ↪as file:
        pred= pickle.load(file)
    data=attn[:,0,:,:].squeeze(0).numpy() #
    numpairs,attr,label=environment
    numpair_numpy=numpairs.squeeze(0).numpy()#.astype(str)
    dataframe=pd.DataFrame(data,index=numpair_numpy,columns=numpair_numpy)

    return attn,numpairs,attr,label,pred


def group_tensors_by_value(tensor_list):
    """
       tensor                 tensor
    :param tensor_list:      tensor
    :return:               tensor
    """
    #       tensor        tensor
    tensor_dict = {}
    for i, tensor in enumerate(tensor_list):
        found = False
        for key in tensor_dict:
            if torch.equal(key, tensor):
                tensor_dict[key].append(i)
                found = True
                break
        if not found:
            tensor_dict[tensor] = [i]

    #              tensor
    grouped_tensors = []
    index_list = []
    for tensor, indices in tensor_dict.items():
        if len(indices) > 1:
            grouped_tensor = torch.stack([tensor_list[i] for i in indices])
            grouped_tensors.append(grouped_tensor)
            index_list.append(indices)

    #
    return grouped_tensors, index_list

def attn_clean(attn,numpairs):
    numpair_numpy=numpairs.squeeze(0).cpu().numpy().astype(str)
    dataframe=pd.DataFrame(attn.squeeze(0)[0,:,:
 ↪],index=numpair_numpy,columns=numpair_numpy).drop(index='0',columns='0')
    return dataframe

def get_data(Ds,select_datas,length):
    manu_seeds = [f"{i}" for i in range(5)]
    txtpaths = ["981762","981808","981814"]
    numpair_list=[open_attn(Ds,i,j)[1] for i in manu_seeds for j in range(length)]
    raw_list=[open_attn(Ds,i,j) for i in manu_seeds for j in range(length)]
    _,index=group_tensors_by_value(numpair_list)
    pred=[raw_list[i][-1] for i in select_datas]
    attr=[raw_list[i][-3] for i in select_datas]
    print(pred,attr)
    datas=[attn_clean(*raw_list[i][0:2]) for i in select_datas]
    return datas

# manu_seeds = [f"manu_seed={i}" for i in range(5)]
# txtpaths = ["981762","981808","981814"]
# g = [[a:1]]
# [g[-1].append(y) if x == y else g.append([y]) for x, y in zip(a[:-1], a[1:])]
```

[147]:

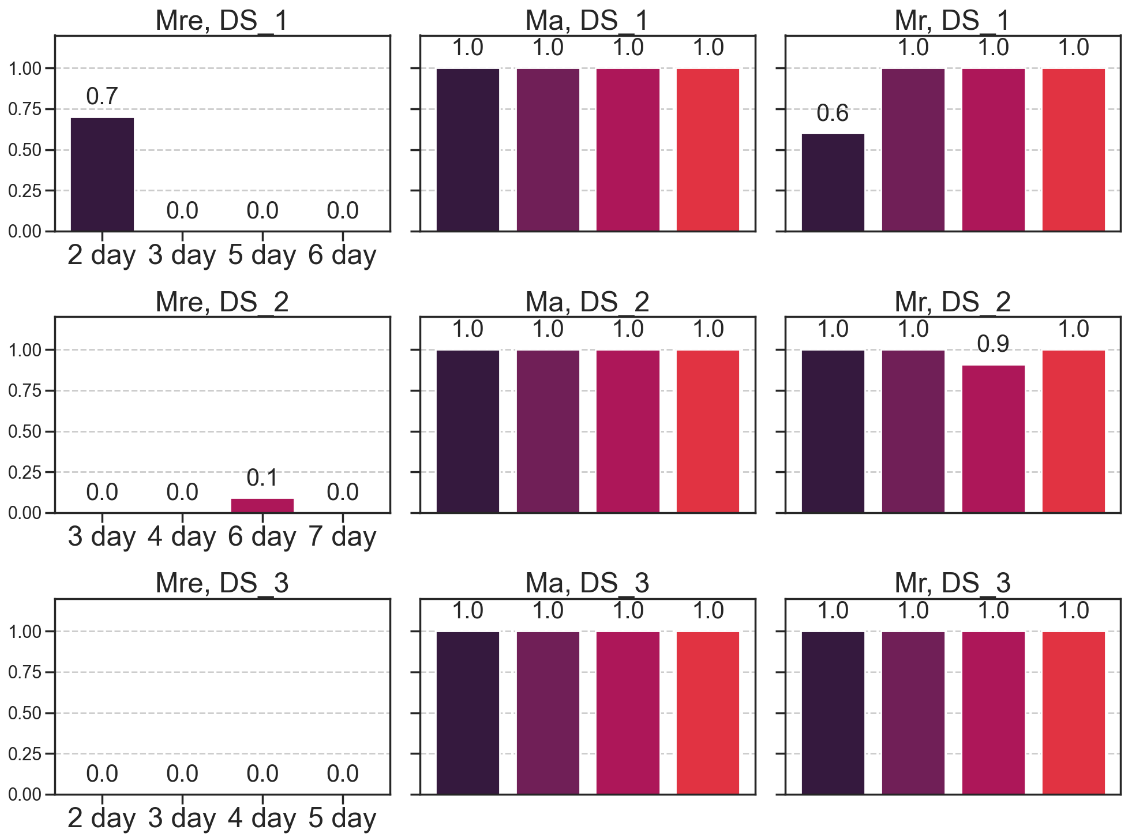
```
# 762  [ 163,136,106,68]   [[0.7, 1.0, 0.6], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]] 2356
# 808 [20, 65, 242, 246]  [[0.0, 1.0, 1.0],[0.0, 1.0, 1.0],[0.09090909090909091, 1.0, 0.9090909090909091],[0.0, 1.0, 1.0]] 3467
# 814 [ 550, 492,152, 182 ]   [[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]],  2345



# 762-->164 808-->70 814-->114
ds1=get_data('981762',[ 163,136,106,68],164)
ds2=get_data('981808',[20, 65, 242, 246],70)
ds3=get_data('981814',[ 550, 492,152, 182 ],114)
datas=ds1+ds2+ds3
```

```
[[0.7, 1.0, 0.6], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]]
[tensor([[[3, 8],
         [4, 7],
         [5, 6],
         [6, 5],
         [0, 4],
         [1, 3],
         [2, 2]]]), tensor([[[3, 9],
         [4, 8],
         [5, 7],
         [6, 6],
         [0, 5],
         [1, 4],
         [2, 3]]]), tensor([[[ 3, 11],
         [ 4, 10],
         [ 5,  9],
         [ 6,  8],
         [ 0,  7],
         [ 1,  6],
         [ 2,  5]]]), tensor([[[ 3, 12],
         [ 4, 11],
         [ 5, 10],
         [ 6,  9],
         [ 0,  8],
         [ 1,  7],
         [ 2,  6]]])]
[[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.09090909090909091, 1.0,
0.9090909090909091], [0.0, 1.0, 1.0]] [tensor([[[3, 9],
         [4, 8],
         [5, 7],
         [6, 6],
         [0, 5],
         [1, 4],
         [2, 3]]]), tensor([[[ 3, 10],
         [ 4,  9],
         [ 5,  8],
         [ 6,  7],
         [ 0,  6],
         [ 1,  5],
         [ 2,  4]]]), tensor([[[ 3, 11],
         [ 4, 10],
         [ 5,  9],
         [ 6,  8],
         [ 0,  7],
         [ 1,  6],
         [ 2,  5]]]), tensor([[[ 3, 12],
         [ 4, 11],
         [ 5, 10],
         [ 6,  9],
         [ 0,  8],
         [ 1,  7],
         [ 2,  6]]])]
[[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]]
[tensor([[[6, 7],
         [0, 6],
         [1, 5],
         [2, 4],
         [3, 3],
         [4, 2],
         [5, 1]]]), tensor([[[6, 8],
         [0, 7],
         [1, 6],
         [2, 5],
         [3, 4],
         [4, 3],
         [5, 2]]]), tensor([[[6, 9],
         [0, 8],
         [1, 7],
         [2, 6],
         [3, 5],
         [4, 4],
         [5, 3]]]), tensor([[[ 6, 11],
         [ 0, 10],
         [ 1,  9],
         [ 2,  8],
         [ 3,  7],
         [ 4,  6],
         [ 5,  5]]])]
```

[149]:
```
data=[[[0.7, 1.0, 0.6], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]],[[0.
 ↪0, 1.0, 1.0],[0.0, 1.0, 1.0],[0.09090909090909091, 1.0, 0.
 ↪9090909090909091],[0.0, 1.0, 1.0]],[[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0,␣
 ↪1.0, 1.0], [0.0, 1.0, 1.0]]]
data
```

[149]:
```
[[[0.7, 1.0, 0.6], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]],
 [[0.0, 1.0, 1.0],
  [0.0, 1.0, 1.0],
  [0.09090909090909091, 1.0, 0.9090909090909091],
  [0.0, 1.0, 1.0]],
 [[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]]]
```

[43]:
```
import matplotlib.pyplot as plt
import numpy as np

data = [
    [[0.7, 1.0, 0.6], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]],
    [[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.09090909090909091, 1.0, 0.
 ↪9090909090909091], [0.0, 1.0, 1.0]],
    [[0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0], [0.0, 1.0, 1.0]]
]

fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(20,15),)
x_labels = [['2 day', '3 day', '5 day', '6 day'], ['3 day', '4 day', '6 day',␣
 ↪'7 day'], ['2 day', '3 day', '4 day', '5 day']]
metrics = ['Mre', 'Ma', 'Mr']
#   'hls'
colors =sns.color_palette("rocket")
# fig.suptitle('Performance Metrics Across Data Sets', fontsize=2)

for i, metric in enumerate(metrics):
    for j in range(3):
        values = [x[i] for x in data[j]]

        axs[j, i].bar(np.arange(len(values)), values,␣
 ↪align='center',color=colors,)
        axs[j, i].set_xticks(np.arange(len(values)))
        axs[j, i].set_xticklabels([f'DS {x+1}' for x in range(len(values))],␣
 ↪fontsize=35)
        axs[j, i].set_ylim([0, 1.2])
        axs[j, i].set_title(f'{metric}, DS_{j+1}', fontsize=35)
        axs[j, i].set_ylabel("")
        axs[j, i].grid(axis='y', linestyle='--')
        for idx, val in enumerate(values):
            axs[j, i].text(idx, val+0.05, f'{val:.1f}', fontsize=30,␣
 ↪ha='center', va='bottom')

        if i == 0:
            axs[j, i].set_xticklabels(x_labels[j], fontsize=35)
#             axs[j, i].set_ylabel(metric, fontsize=20)
        else:
            axs[j, i].set_xticks([])
            axs[j, i].set_yticklabels([])

plt.tight_layout()
plt.savefig('pdf\\attn2.pdf', dpi=600)
plt.show()
```



2.1 , ,

,

3 ,

[ ]:

```
import pandas as pd
import numpy as np
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
dir="981814.txt"
lines = open('lstm_attn\\     \\'+dir, encoding='utf-8').read().strip().
  ↪split('\n')
#
pairs = [[s for s in l.split('#')] for l in lines]
numpair=[]      #
attr=[]         #
traj_day_len=[]#
label=[]
for week_rank in range(len(pairs)):
    label+=[pairs[week_rank][7].split(',')]
    for day in range(7):
        numpair+=[pairs[week_rank][day].split('Attr')[0].split(',')[:-1]]
        traj_day_len.append(len(pairs[week_rank][day].split('Attr')[0].
 ↪split(',')[:-1]))
        attr.append([[pairs[week_rank][day].split('Attr')[1].split(',')[1:]]])
nested_list = [i[:7] for i in  pairs[::49]]
flat_list = [item for sublist in nested_list for item in (sublist if␣
 ↪isinstance(sublist, list) else [sublist])]
my_list = flat_list
new_list = []

for sub_list in my_list:
    sub_list = sub_list.split(",")[:-3]
    sub_list = [x for x in sub_list if x != "Attr"]
    new_list.append(",".join(sub_list))

new_dict = {}

for index, item in enumerate(new_list):
    new_dict[index] = item.split(",")
data =new_dict

df = pd.DataFrame(columns=['key','value'])
for key in data:
    for item in data[key]:
        df = df.append({'key':key,'value':item},ignore_index=True)
df[['lon','lat']] = df['value'].str.split('P', expand=True).astype(int)/100
df.to_excel('output.xlsx', index=False)
df['end_lat'] = df['lat'].shift(-1)
df['end_lon'] = df['lon'].shift(-1)
indexes_to_drop = []

#
for i in range(1, len(df)):
    #  key
    if df.loc[i, 'key'] != df.loc[i-1, 'key']:
        indexes_to_drop.append(i-1)
#
df = df.drop(indexes_to_drop)
#
df = df.reset_index(drop=True)
```

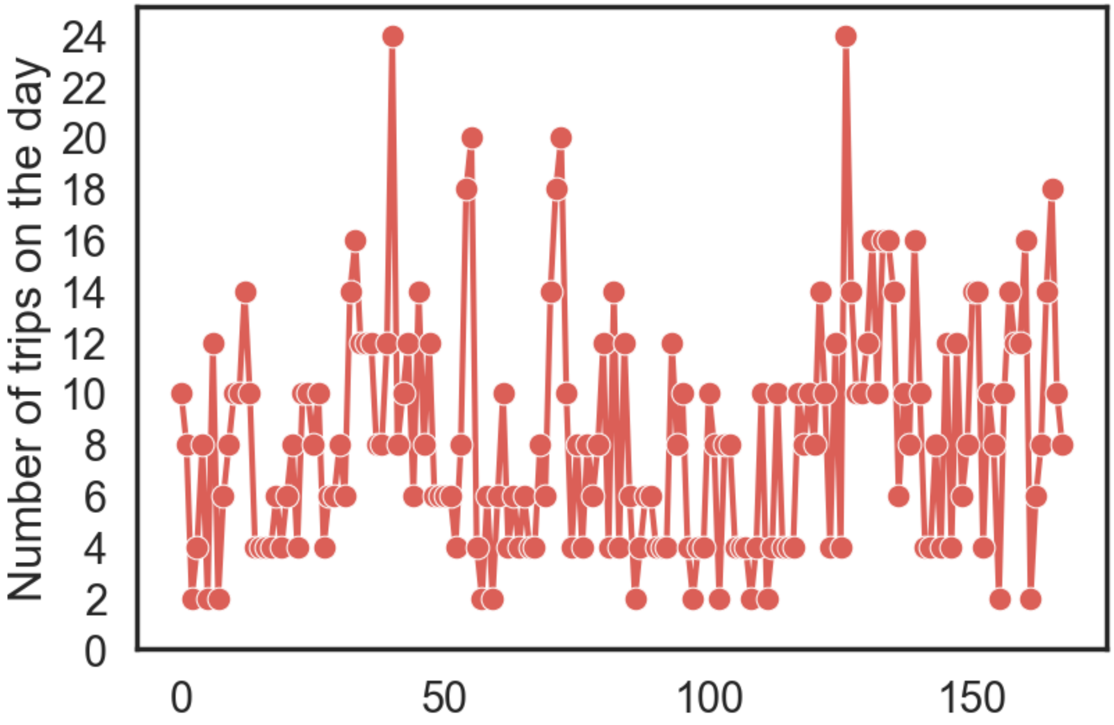
3.0.1

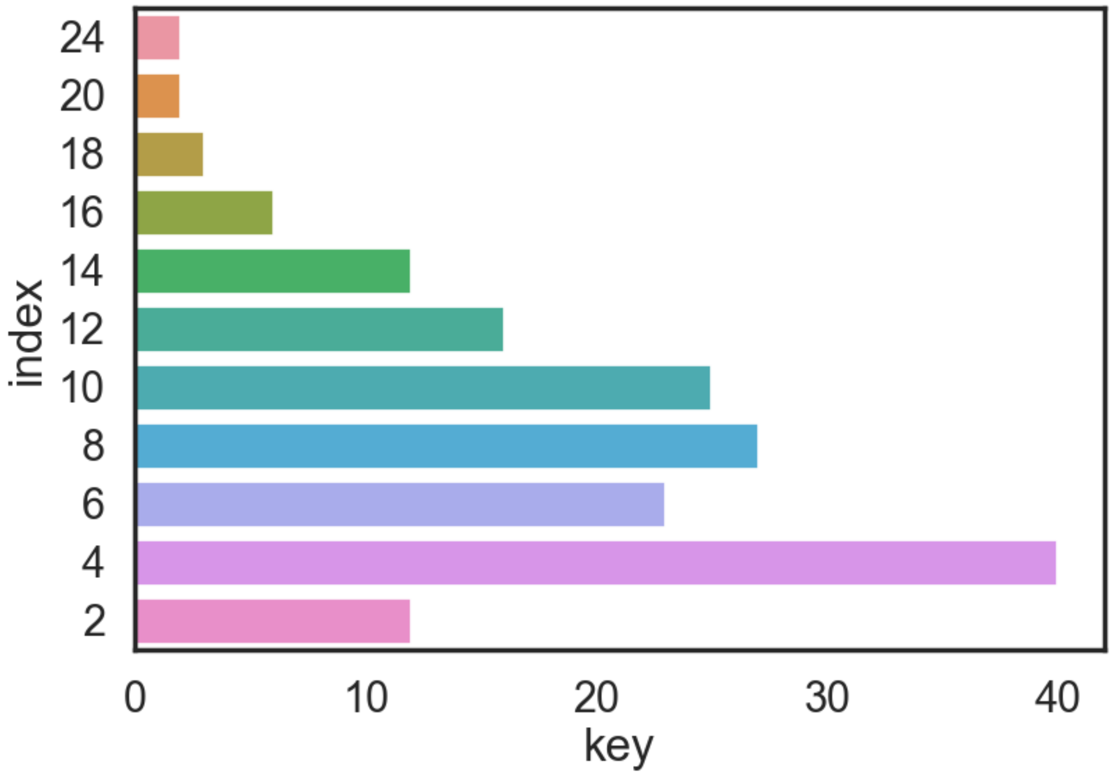
```
[7]: dir="981762.txt"
     # dir[0:7]
     a=f'{dir[0:7]}'
     a
```

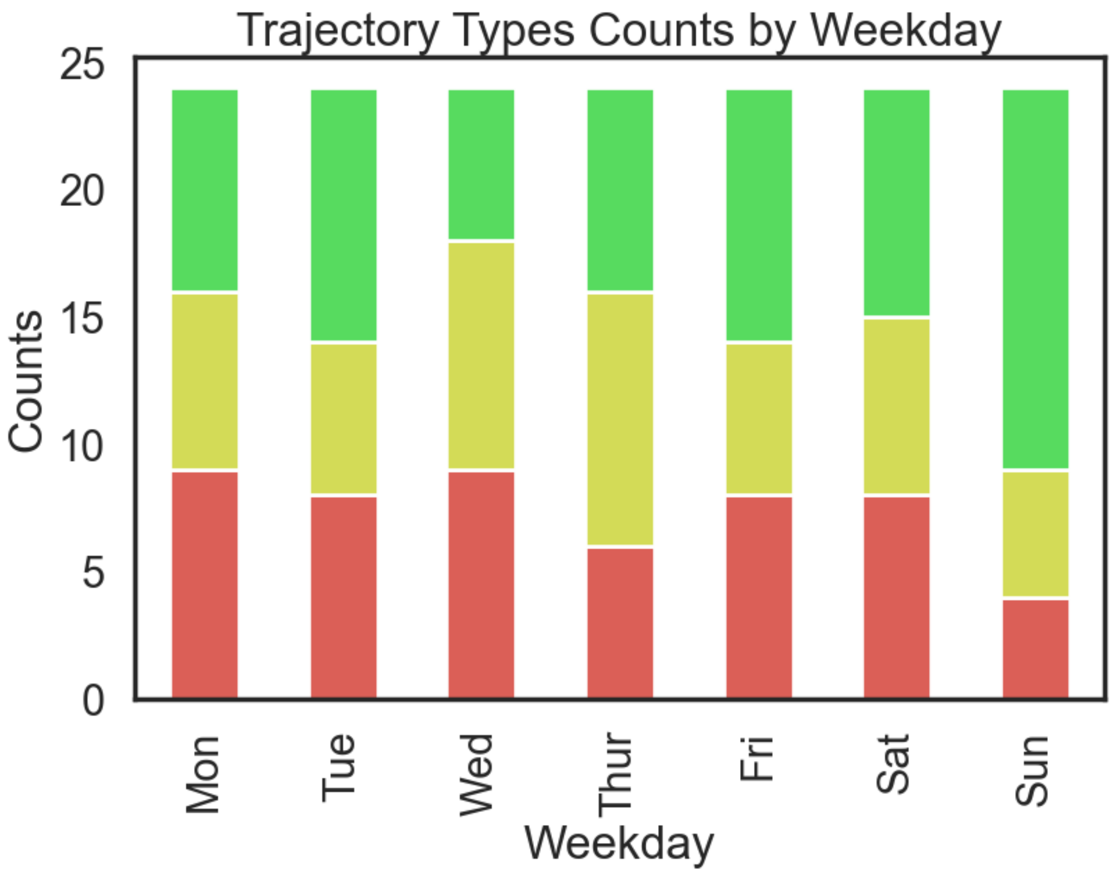
```
[7]: '981762.'
```

```
[47]: import pandas as pd
      import numpy as np
      import warnings
      warnings.simplefilter(action='ignore', category=FutureWarning)
      dir="981814.txt"
      lines = open('lstm_attn\\    \\'+dir, encoding='utf-8').read().strip().
        ↪split('\n')
      #
      pairs = [[s for s in l.split('#')] for l in lines]
      numpair=[]     #
      attr=[]       #
      traj_day_len=[]#
      label=[]
      for week_rank in range(len(pairs)):
          label+=[pairs[week_rank][7].split(',')]
          for day in range(7):
              numpair+=[pairs[week_rank][day].split('Attr')[0].split(',')[:-1]]
              traj_day_len.append(len(pairs[week_rank][day].split('Attr')[0].
        ↪split(',')[:-1]))
              attr.append([[pairs[week_rank][day].split('Attr')[1].split(',')[1:]]])
      nested_list = [i[:7] for i in  pairs[::49]]
      flat_list = [item for sublist in nested_list for item in (sublist if␣
        ↪isinstance(sublist, list) else [sublist])]
      my_list = flat_list
      new_list = []

      for sub_list in my_list:
          sub_list = sub_list.split(",")[:-3]
          sub_list = [x for x in sub_list if x != "Attr"]
          new_list.append(",".join(sub_list))

      new_dict = {}

      for index, item in enumerate(new_list):
          new_dict[index] = item.split(",")
      data =new_dict

      df = pd.DataFrame(columns=['key','value'])
```

```
for key in data:
    for item in data[key]:
        df = df.append({'key':key,'value':item},ignore_index=True)
df[['lon','lat']] = df['value'].str.split('P', expand=True).astype(int)/100
df.to_excel('output.xlsx', index=False)
df['end_lat'] = df['lat'].shift(-1)
df['end_lon'] = df['lon'].shift(-1)
date_counts = df['key'].value_counts()
#
date_counts.sort_index(inplace=True)
sns.set_context("poster")
sns.set_palette('hls')
plt.figure(figsize=(9, 6))
sns.lineplot(data=date_counts, marker='o', legend=True)
plt.ylabel('Number of trips on the day')
plt.yticks(np.arange(0, max(date_counts)+1, 2))



plt.savefig(f'pdf\{dir[0:7]}_a.pdf', dpi=600)
plt.show()


count_len_traj=pd.DataFrame(date_counts.value_counts()).reset_index().
 ↪sort_values('index',ascending=False)
plt.figure(figsize=(9, 6))
sns.set_context("poster")
sns.barplot(x='key', y='index',␣
 ↪data=count_len_traj,order=count_len_traj['index'].values, orient='h')
plt.savefig(f'pdf\{dir[0:7]}_b.pdf', dpi=600)

key_counts = df.groupby('key').size().reset_index(name='count')
key_counts['type'] = ''
key_counts.loc[(key_counts['count']>0) & (key_counts['count']<6), 'type'] =␣
 ↪'duan'
key_counts.loc[(key_counts['count']>5) & (key_counts['count']<10), 'type'] =␣
 ↪'zhong'
key_counts.loc[(key_counts['count']>9), 'type'] = 'chang'

# key_counts  week
key_counts['week'] = key_counts['key'] % 7
#  week
key_counts['week'] = key_counts['week'].replace({0:'Tue',1:'Wed',2:'Thur',3:
 ↪'Fri',4:'Sat',5:'Sun',6:'Mon'})
#
type_counts = pd.crosstab(key_counts['week'], key_counts['type'])
type_counts = type_counts.reindex(['Mon','Tue','Wed','Thur','Fri','Sat','Sun'])
type_counts = type_counts[['duan', 'zhong', 'chang']]

# 3
sns.set_style("white")
sns.set_context("poster")
sns.set_palette('hls')
type_counts.plot(kind='bar', stacked=True, alpha=1, figsize=(9, 6),␣
 ↪legend=False)
plt.xlabel('Weekday')
plt.ylabel('Counts')
plt.title('Trajectory Types Counts by Weekday')
plt.gca().yaxis.grid(False)
plt.savefig(f'pdf\{dir[0:7]}_c.pdf', dpi=600)
plt.show()
```




24
20
18
16
14
12
10
8
6
4
2
index
0
10
20
30
40
key



[49]:

```
import pandas as pd
#
indexes_to_drop = []

#
for i in range(1, len(df)):
    #  key
    if df.loc[i, 'key'] != df.loc[i-1, 'key']:
        indexes_to_drop.append(i-1)
#
df = df.drop(indexes_to_drop)
#
df = df.reset_index(drop=True)
df
```

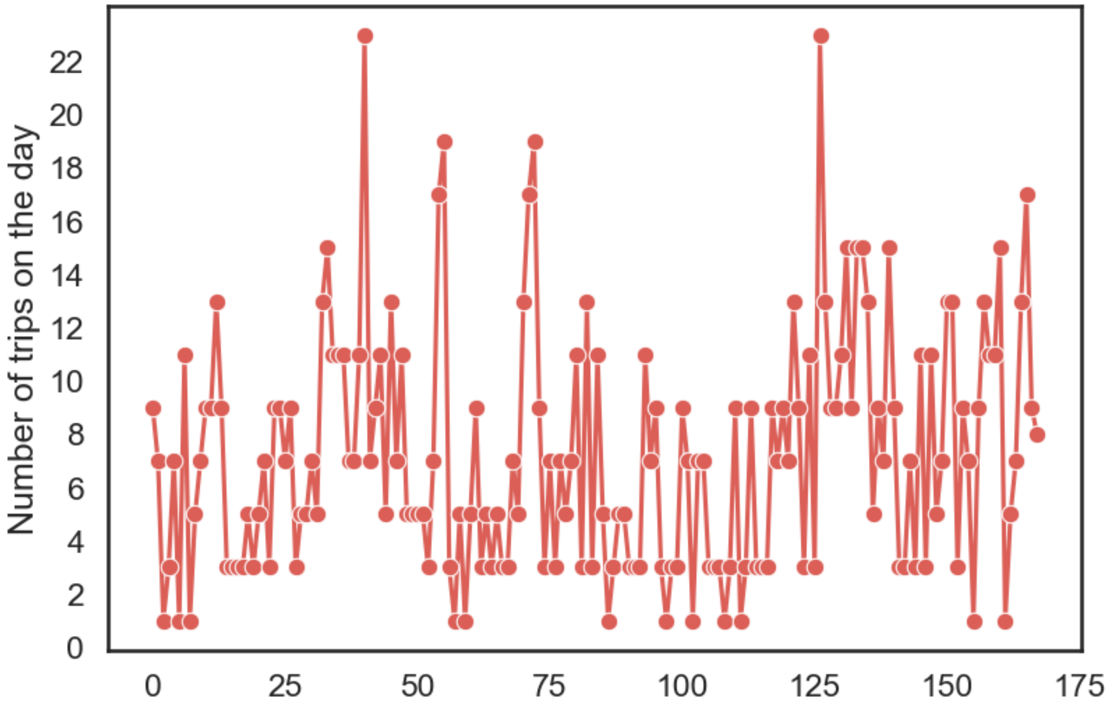
[49]:

```
     key       value     lon    lat  end_lat  end_lon
0      0  11896P3063  118.96  30.63    30.62   118.97
1      0  11897P3062  118.97  30.62    30.62   118.97
2      0  11897P3062  118.97  30.62    30.63   118.98
```

```
3       0  11898P3063  118.98  30.63    30.63   118.98
4       0  11898P3063  118.98  30.63    30.62   118.97
…    …         …      …     …      …       …
1214  167  12145P3132  121.45  31.32    31.32   121.45
1215  167  12145P3132  121.45  31.32    31.33   121.44
1216  167  12144P3133  121.44  31.33    31.33   121.44
1217  167  12144P3133  121.44  31.33    31.33   121.43
1218  167  12143P3133  121.43  31.33      NaN      NaN

[1219 rows x 6 columns]
```

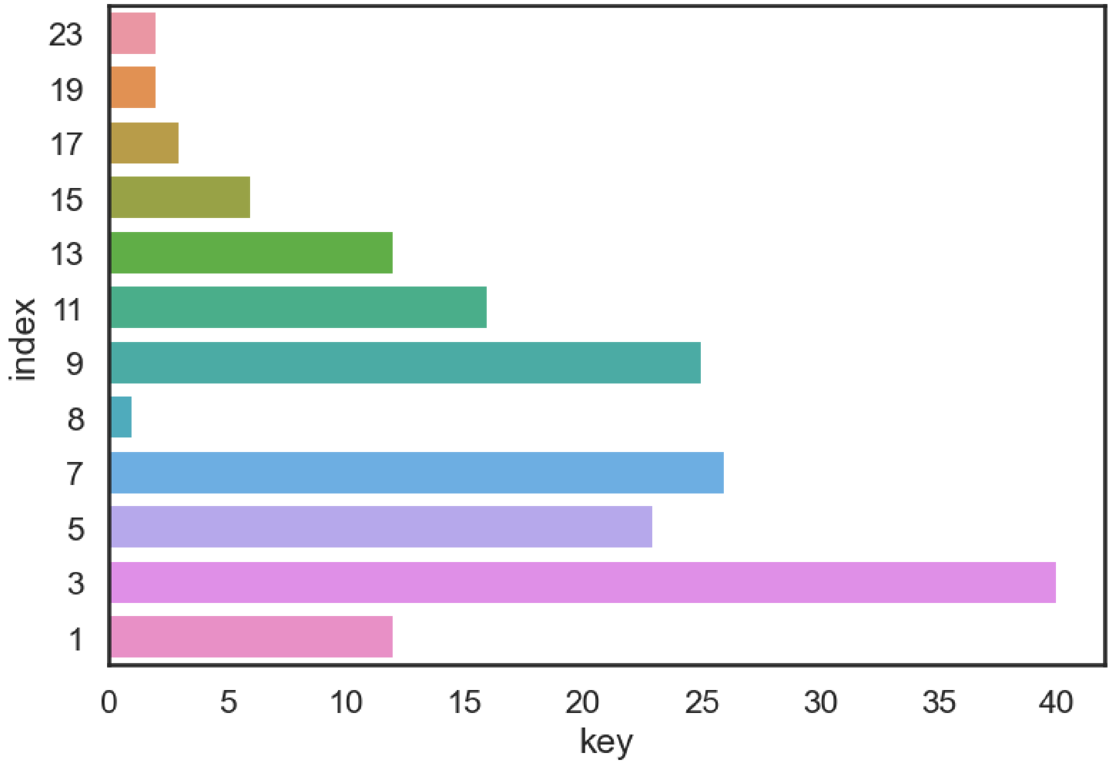
[50]:

```python
date_counts = df['key'].value_counts()
#
date_counts.sort_index(inplace=True)
sns.set_palette('hls')
plt.figure(figsize=(9, 6))
sns.lineplot(data=date_counts, marker='o', legend=True)
plt.ylabel('Number of trips on the day')
plt.yticks(np.arange(0, max(date_counts)+1, 2))
plt.savefig('eps\summ_33.eps', dpi=600, format='eps')
plt.show()
```



[51]:

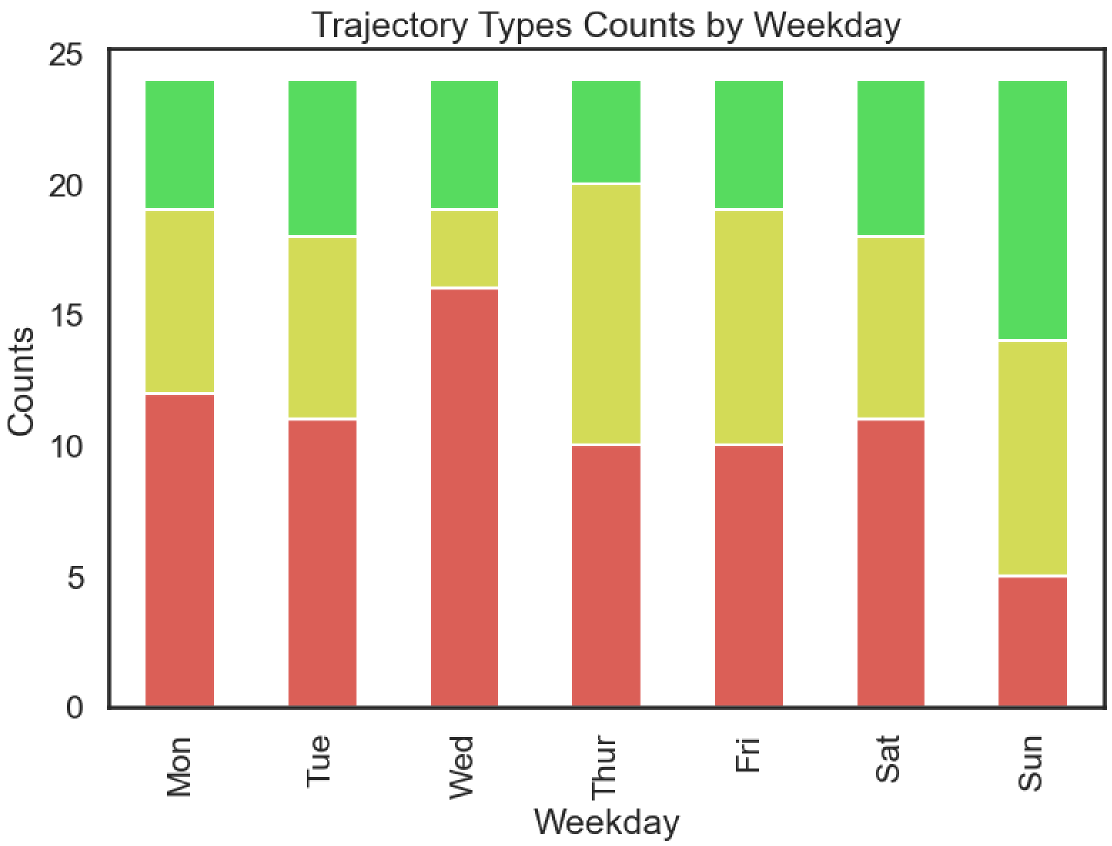
```
count_len_traj=pd.DataFrame(date_counts.value_counts()).reset_index().
  ↪sort_values('index',ascending=False)
plt.figure(figsize=(9, 6))
sns.barplot(x='key', y='index',␣
  ↪data=count_len_traj,order=count_len_traj['index'].values, orient='h')
plt.savefig('eps\summ_34.eps', dpi=600, format='eps')
```



[52]:

```
#       key
key_counts = df.groupby('key').size().reset_index(name='count')
key_counts['type'] = ''
key_counts.loc[(key_counts['count']>0) & (key_counts['count']<6), 'type'] =␣
  ↪'duan'
key_counts.loc[(key_counts['count']>5) & (key_counts['count']<10), 'type'] =␣
  ↪'zhong'
key_counts.loc[(key_counts['count']>9), 'type'] = 'chang'

#key_counts  week
key_counts['week'] = key_counts['key'] % 7
#  week
key_counts['week'] = key_counts['week'].replace({0:'Tue',1:'Wed',2:'Thur',3:
  ↪'Fri',4:'Sat',5:'Sun',6:'Mon'})
#
```

```
type_counts = pd.crosstab(key_counts['week'], key_counts['type'])
type_counts = type_counts.reindex(['Mon','Tue','Wed','Thur','Fri','Sat','Sun'])
type_counts = type_counts[['duan', 'zhong', 'chang']]

# 3
sns.set_style("white")
sns.set_palette('hls')
type_counts.plot(kind='bar', stacked=True, alpha=1, figsize=(9, 6),␣
 ↪legend=False)
plt.xlabel('Weekday')
plt.ylabel('Counts')
plt.title('Trajectory Types Counts by Weekday')
plt.gca().yaxis.grid(False)
plt.savefig('eps\summ_31.eps', dpi=600, format='eps')
plt.show()
```



3.1

[102]:

```
import pandas as pd
import numpy as np
dir="981814.txt"
lines = open('lstm_attn\\    \\'+dir, encoding='utf-8').read().strip().
  ↪split('\n')
#
pairs = [[s for s in l.split('#')] for l in lines]
numpair=[]     #
attr=[]        #
traj_day_len=[]#
label=[]
for week_rank in range(len(pairs)):
    label+=[pairs[week_rank][7].split(',')]
    for day in range(7):
        numpair+=[pairs[week_rank][day].split('Attr')[0].split(',')[:-1]]
        traj_day_len.append(len(pairs[week_rank][day].split('Attr')[0].
  ↪split(',')[:-1]))
        attr.append([[pairs[week_rank][day].split('Attr')[1].split(',')[1:]]])
nested_list = [i[:7] for i in  pairs[::49]]
flat_list = [item for sublist in nested_list for item in (sublist if␣
  ↪isinstance(sublist, list) else [sublist])]
my_list = flat_list
new_list = []

for sub_list in my_list:
    sub_list = sub_list.split(",")[:-3]
    sub_list = [x for x in sub_list if x != "Attr"]
    new_list.append(",".join(sub_list))

new_dict = {}

for index, item in enumerate(new_list):
    new_dict[index] = item.split(",")
data =new_dict

df = pd.DataFrame(columns=['key','value'])
for key in data:
    for item in data[key]:
        df = df.append({'key':key,'value':item},ignore_index=True)
df[['lon','lat']] = df['value'].str.split('P', expand=True).astype(int)/100
df.to_excel('output.xlsx', index=False)
df['end_lat'] = df['lat'].shift(-1)
df['end_lon'] = df['lon'].shift(-1)
#
df = df[(df['lat'] != df['end_lat']) | (df['lon'] != df['end_lon'])]
df = df[:-1].rename(columns={ "key": "id", "lon": "lng_h", "lat": "lat_h",␣
 ↪"end_lat": "lat_w", "end_lon": "lng_w"})
df=pd.DataFrame(dict(
    COORDINATES=list(zip(df['lng_h'],df['lat_h']))
            )
            )
df['SPACES']=1
```

[103]:

```
import pydeck as pdk
import pandas as pd

layer = pdk.Layer(
    "ScreenGridLayer",
    df,
    pickable=False,
    opacity=0.8,
    cell_size_pixels=60,
#     color_range=[
#         [0, 25, 0, 25],
#         [0, 85, 0, 85],
#         [0, 127, 0, 127],
#         [0, 170, 0, 170],
#         [0, 190, 0, 190],
#         [0, 255, 0, 255],
#     ],#
    get_position="COORDINATES",
    get_weight="SPACES",
)
# Set the viewport location
view_state = pdk.ViewState(latitude=df.iloc[0,0][1], longitude=df.iloc[0,0][0],␣
 ↪zoom=12, bearing=0, pitch=0)
# Render
r = pdk.
 ↪Deck(layers=[layer],initial_view_state=view_state,map_style='light_no_labels')#


r.to_html("eps\summ_32.html")
```

[103]:

```
<IPython.core.display.HTML object>
```

4

4.1 1

[30]:

```
import pandas as pd
df=pd.read_excel("6 3  .xlsx",sheet_name=['data1','data2'])['data2']
raw_data=df.melt(['model','attr','DS'],['MRE','MA','MR'],'index','value')
raw=raw_data[raw_data['attr']=='mean']
raw
```

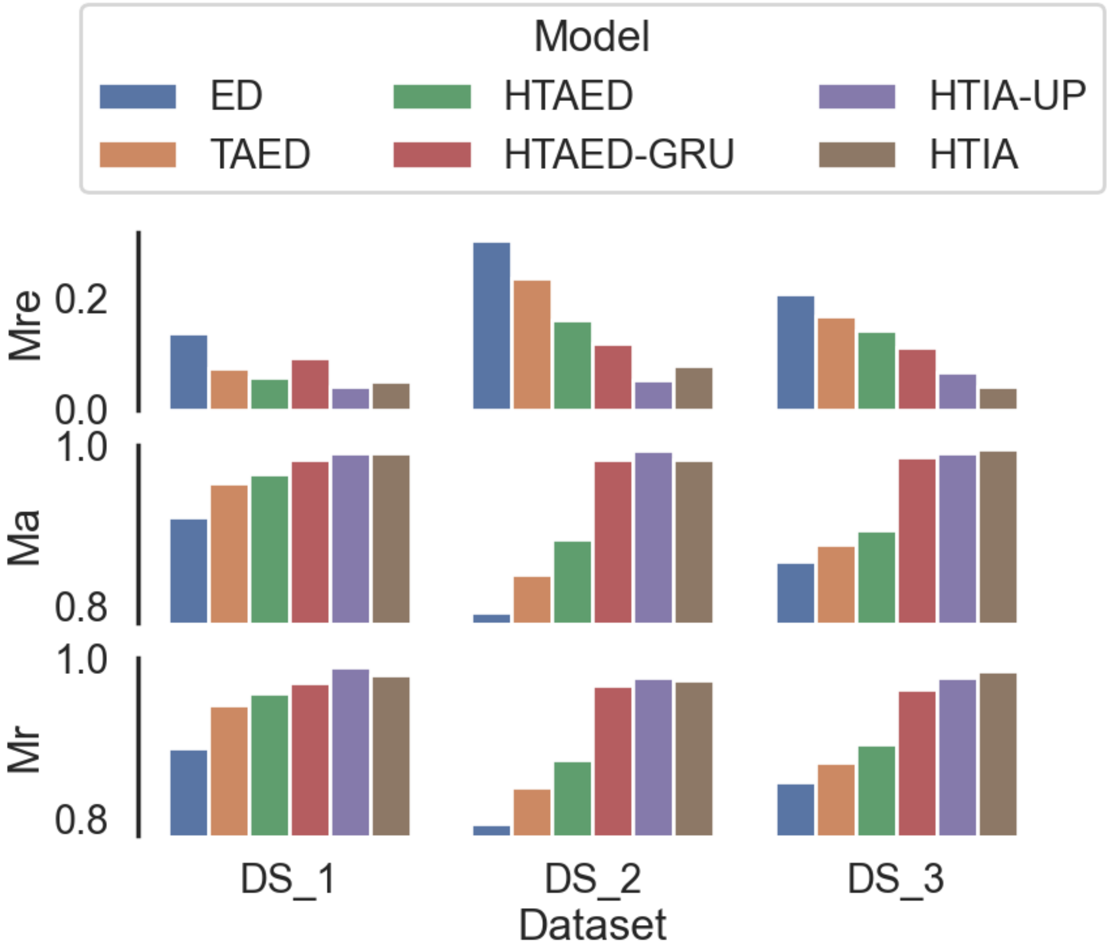
[30]:

```
          model  attr    DS index     value
5            ED  mean  DS_1   MRE  0.137000
6          TAED  mean  DS_1   MRE  0.075000
7         HTAED  mean  DS_1   MRE  0.059000
8     HTAED-GRU  mean  DS_1   MRE  0.092806
9       HTIA-UP  mean  DS_1   MRE  0.040763
20           ED  mean  DS_2   MRE  0.303000
21         TAED  mean  DS_2   MRE  0.236000
22        HTAED  mean  DS_2   MRE  0.161000
23    HTAED-GRU  mean  DS_2   MRE  0.118348
24      HTIA-UP  mean  DS_2   MRE  0.052700
35           ED  mean  DS_3   MRE  0.207000
36         TAED  mean  DS_3   MRE  0.169000
37        HTAED  mean  DS_3   MRE  0.143000
38    HTAED-GRU  mean  DS_3   MRE  0.112051
39      HTIA-UP  mean  DS_3   MRE  0.068234
46         HTIA  mean  DS_1   MRE  0.051323
49         HTIA  mean  DS_2   MRE  0.078699
52         HTIA  mean  DS_3   MRE  0.041432
59           ED  mean  DS_1    MA  0.911000
60         TAED  mean  DS_1    MA  0.953000
61        HTAED  mean  DS_1    MA  0.965000
62    HTAED-GRU  mean  DS_1    MA  0.981757
63      HTIA-UP  mean  DS_1    MA  0.990884
74           ED  mean  DS_2    MA  0.792000
75         TAED  mean  DS_2    MA  0.839000
76        HTAED  mean  DS_2    MA  0.883000
77    HTAED-GRU  mean  DS_2    MA  0.981560
78      HTIA-UP  mean  DS_2    MA  0.992982
89           ED  mean  DS_3    MA  0.856000
90         TAED  mean  DS_3    MA  0.877000
91        HTAED  mean  DS_3    MA  0.895000
92    HTAED-GRU  mean  DS_3    MA  0.985876
93      HTIA-UP  mean  DS_3    MA  0.989886
100        HTIA  mean  DS_1    MA  0.990302
103        HTIA  mean  DS_2    MA  0.981416
106        HTIA  mean  DS_3    MA  0.995201
113          ED  mean  DS_1    MR  0.889000
```

```
114       TAED  mean  DS_1    MR  0.942000
115      HTAED  mean  DS_1    MR  0.956000
116  HTAED-GRU  mean  DS_1    MR  0.969419
117    HTIA-UP  mean  DS_1    MR  0.988327
128         ED  mean  DS_2    MR  0.794000
129       TAED  mean  DS_2    MR  0.840000
130      HTAED  mean  DS_2    MR  0.874000
131  HTAED-GRU  mean  DS_2    MR  0.965682
132    HTIA-UP  mean  DS_2    MR  0.975431
143         ED  mean  DS_3    MR  0.846000
144       TAED  mean  DS_3    MR  0.871000
145      HTAED  mean  DS_3    MR  0.894000
146  HTAED-GRU  mean  DS_3    MR  0.960650
147    HTIA-UP  mean  DS_3    MR  0.976155
154       HTIA  mean  DS_1    MR  0.979661
157       HTIA  mean  DS_2    MR  0.972718
160       HTIA  mean  DS_3    MR  0.984073
```

[31]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="white", context="poster")
# Set up the matplotlib figure
f, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(9, 6), sharex=True)
sns.barplot(data=raw.query('index=="MRE" '),x='DS',y='value',hue='model',ax=ax1)
sns.barplot(data=raw[raw['index']=='MA'],x='DS',y='value',hue='model',ax=ax2)
ax2.set_ylim([0.78, 1])
sns.barplot(data=raw[raw['index']=='MR'],x='DS',y='value',hue='model',ax=ax3)
ax3.set_ylim([0.78, 1])
ax1.legend_.remove()
ax2.legend_.remove()
ax3.legend_.remove()
ax1.set_xlabel('')
ax2.set_xlabel('')
ax1.set_ylabel('Mre')
ax2.set_ylabel('Ma')
ax3.set_ylabel('Mr')
plt.legend(title='Model', loc='upper center', bbox_to_anchor=(0.5,4.8), ncol=3)
plt.xlabel('Dataset')

sns.despine(bottom=True)
plt.savefig('pdf\\result1.pdf', dpi=300,bbox_inches = 'tight')
# plt.setp(f.axes, yticks=[])
# plt.tight_layout(h_pad=2)
```



4.2

```
[128]: import pandas as pd
       df=pd.read_excel("6 3  .xlsx",sheet_name=['data1','data2'])['data1']
       raw_data=df.melt(['model','attr','DS'],['MRE','MA','MR'],'index','value')
       raw=raw_data[raw_data['attr']=='mean']
       raw
```

```
[128]:              model  attr    DS index     value
       3            HTIA  mean  DS_1   MRE  0.051323
       4         HTIA-UP  mean  DS_1   MRE  0.040763
       5   T-Transformer  mean  DS_1   MRE  0.069890
       12           HTIA  mean  DS_2   MRE  0.078699
       13        HTIA-UP  mean  DS_2   MRE  0.052700
       14  T-Transformer  mean  DS_2   MRE  0.035963
       21           HTIA  mean  DS_3   MRE  0.041432
       22        HTIA-UP  mean  DS_3   MRE  0.068234
```

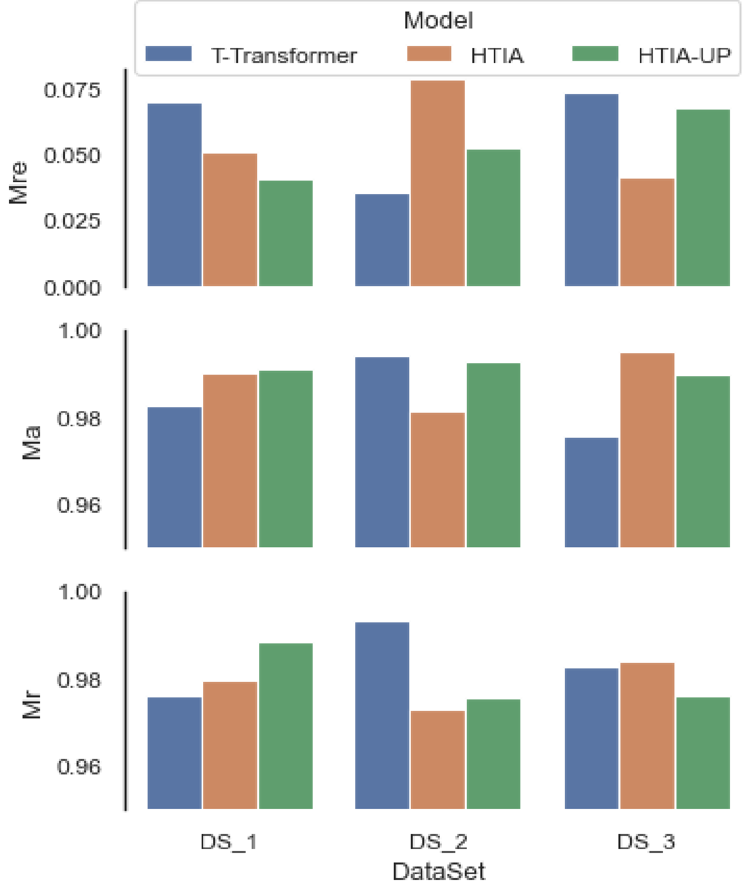
```
23  T-Transformer  mean  DS_3   MRE  0.073580
30           HTIA  mean  DS_1    MA  0.990302
31        HTIA-UP  mean  DS_1    MA  0.990884
32  T-Transformer  mean  DS_1    MA  0.982825
39           HTIA  mean  DS_2    MA  0.981416
40        HTIA-UP  mean  DS_2    MA  0.992982
41  T-Transformer  mean  DS_2    MA  0.994026
48           HTIA  mean  DS_3    MA  0.995201
49        HTIA-UP  mean  DS_3    MA  0.989886
50  T-Transformer  mean  DS_3    MA  0.975459
57           HTIA  mean  DS_1    MR  0.979661
58        HTIA-UP  mean  DS_1    MR  0.988327
59  T-Transformer  mean  DS_1    MR  0.975898
66           HTIA  mean  DS_2    MR  0.972718
67        HTIA-UP  mean  DS_2    MR  0.975431
68  T-Transformer  mean  DS_2    MR  0.993379
75           HTIA  mean  DS_3    MR  0.984073
76        HTIA-UP  mean  DS_3    MR  0.976155
77  T-Transformer  mean  DS_3    MR  0.982795
```

[129]:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="white", context="paper")
# Set up the matplotlib figure
f, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(4.2,5), sharex=True)
hue_order=['T-Transformer', 'HTIA', 'HTIA-UP']
sns.barplot(data=raw.query('index=="MRE"␣
  ↪'),x='DS',y='value',hue='model',hue_order=hue_order,ax=ax1)
sns.
  ↪barplot(data=raw[raw['index']=='MA'],x='DS',y='value',hue='model',hue_order=hue_order,ax=ax
ax2.set_ylim([0.95, 1])
sns.
  ↪barplot(data=raw[raw['index']=='MR'],x='DS',y='value',hue='model',hue_order=hue_order,ax=ax
ax3.set_ylim([0.95, 1])
ax1.legend_.remove()
ax2.legend_.remove()
ax3.legend_.remove()
ax1.set_xlabel('')
ax2.set_xlabel('')
ax1.set_ylabel('Mre')
ax2.set_ylabel('Ma')
ax3.set_ylabel('Mr')
plt.legend(title='Model', loc='lower center', bbox_to_anchor=(0.5, 3.32),␣
  ↪ncol=3)
plt.xlabel('DataSet')

sns.despine(bottom=True)
```

```
# plt.setp(f.axes, yticks=[])
# plt.tight_layout(h_pad=2)
plt.savefig('eps\\result2.eps', dpi=600, format='eps')
```

```
The PostScript backend does not support transparency; partially transparent
artists will be rendered opaque.
```



[ ]:

```
```