ADNI Data Fusion CDR

Step 0. Load

0.1 load packages

In [1]:
```
import sys
import os
import numpy as np          #1.26.3
import nibabel as nib       #5.2.0
import torch                #2.1.2
import pandas as pd         #2.2.1
import psutil
import sklearn
import pandas as pd

import resource
import time
import scipy
import random
import matplotlib.pyplot as plt
import matplotlib.cm as cm  # Colormap module

import SupBigFLICA_cpu10July2024

from SupBigFLICA_cpu10July2024 import grid_search_elasticnet, ElasticNetRegr
```

In [2]:
```
# # Set memory limit (in bytes, 1 GB here)
memory_limit = 50 * 1024 * 1024 * 1024  # 24 GB
resource.setrlimit(resource.RLIMIT_AS, (memory_limit, memory_limit))


# Get current memory limits
soft_limit, hard_limit = resource.getrlimit(resource.RLIMIT_AS)
print(f"Soft limit: {soft_limit / (1024**2)} MB")
print(f"Hard limit: {hard_limit / (1024**2)} MB")



# Set the number of threads to use for BLAS, OpenMP, or other parallel libra
os.environ["OMP_NUM_THREADS"] = "2"  # Limits OpenMP to 4 threads
os.environ["MKL_NUM_THREADS"] = "2"  # Limits Intel MKL to 4 threads
os.environ["OPENBLAS_NUM_THREADS"] = "2"  # Limits OpenBLAS to 4 threads
os.environ["NUMEXPR_NUM_THREADS"] = "2"  # Limits NumExpr to 4 threads
os.environ["VECLIB_MAXIMUM_THREADS"] = "24"  # Limits Apple's Accelerate to
os.environ["MAX_NUM_THREADS"] = "2"  # Generic limit

# Optional: Ensure libraries like NumPy respect the thread limit
np.seterr(all='ignore')
```

```
# Limit PyTorch to 4 threads
torch.set_num_threads(2)
torch.set_num_interop_threads(2)

# Get the current process
process = psutil.Process(os.getpid())

process.cpu_affinity([0, 1, 2])  # Restrict to cores 0 and 1

# # Inspect all threads
threads = process.threads()
print(f"Number of threads: {len(threads)}")
print("Thread details:")
for t in threads:
    print(f"Thread ID: {t.id}, User Time: {t.user_time}, System Time: {t.sys

# Check memory usage
print(f"Memory usage: {process.memory_info().rss / 1e6} MB")  # Resident Set

# Check the number of threads
print(f"Number of threads: {process.num_threads()}")

# Check CPU affinity (cores the process is allowed to run on)
print(f"CPU affinity: {process.cpu_affinity()}")

sys.path.append('/home/ycheng23/SuperBigFLICA_python/Lastest_Versions_14Nov2

sys.version
```

```
Soft limit: 51200.0 MB
Hard limit: 51200.0 MB
Number of threads: 12
Thread details:
Thread ID: 989152, User Time: 2.29, System Time: 0.44
Thread ID: 989170, User Time: 0.0, System Time: 0.0
Thread ID: 989171, User Time: 0.0, System Time: 0.0
Thread ID: 989172, User Time: 0.0, System Time: 0.0
Thread ID: 989173, User Time: 0.0, System Time: 0.0
Thread ID: 989174, User Time: 0.0, System Time: 0.0
Thread ID: 989175, User Time: 0.0, System Time: 0.0
Thread ID: 989176, User Time: 0.0, System Time: 0.0
Thread ID: 989177, User Time: 0.0, System Time: 0.0
Thread ID: 989178, User Time: 0.0, System Time: 0.0
Thread ID: 989180, User Time: 0.0, System Time: 0.0
Thread ID: 989572, User Time: 0.01, System Time: 0.0
Memory usage: 583.675904 MB
Number of threads: 12
CPU affinity: [0, 1, 2]
```

Out[2]: '3.11.9 | packaged by conda-forge | (main, Apr 19 2024, 18:36:13) [GCC 12.3.0]'

0.2 Load data

In [3]:
```
# labels

# nIDPs_validation=np.loadtxt('/data/qnilab/AD_NPS_R01_2022/HCP_Data_Fusion/
# nIDPs_test=np.loadtxt('/data/qnilab/AD_NPS_R01_2022/ADNI_Data_Fusion/Flica
# nIDPs_train=np.loadtxt('/data/qnilab/AD_NPS_R01_2022/ADNI_Data_Fusion/Flic

nIDPs_test=np.genfromtxt('/data/qnilab/AD_NPS_R01_2022/ADNI_Data_Fusion/Flic
    filling_values=np.nan)
nIDPs_train=np.genfromtxt('/data/qnilab/AD_NPS_R01_2022/ADNI_Data_Fusion/Fli
    filling_values=np.nan)


nIDPs_test_npy=np.matrix(nIDPs_test)
nIDPs_train_npy=np.matrix(nIDPs_train)
# nIDPs_validation_npy=np.matrix(nIDPs_validation)

# Change the column indexes to pick out the variables for training (e.g., ir

indices_to_include = 0 #[0, 23] #[19, 20] #[18, 19, 20, 21] # [0, 2, 3, 4, 5

nIDPs_test_targets = nIDPs_test_npy[:, indices_to_include] #[:,list(range(6,
nIDPs_train_set = nIDPs_train_npy[:, indices_to_include] #[:,list(range(6,18
# nIDPs_validation_targets = nIDPs_validation_npy[:,13]
```

In [4]:
```
# features

# root_path = "/data/qnilab/AD_NPS_R01_2022/HCP_Data_Fusion/SBF_Outputs/SBF_
root_path = "/data/qnilab/AD_NPS_R01_2022/ADNI_Data_Fusion/SBF_Outputs/SBF_A
# root_path = "/data/qnilab/AD_NPS_R01_2022/HCP_Data_Fusion/SBF_Outputs/SBF_


lat_train_path = root_path + "/lat_train.csv"
lat_train = pd.read_csv(lat_train_path, header=None)

lat_test_path = root_path + "/lat_test.csv"
lat_test = pd.read_csv(lat_test_path, header=None)

# Convert DataFrames to NumPy arrays
lat_train = lat_train.to_numpy()
lat_test = lat_test.to_numpy()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# lat_train, lat_test, lat_mean, lat_std = standardize_latents(lat_train, la
```

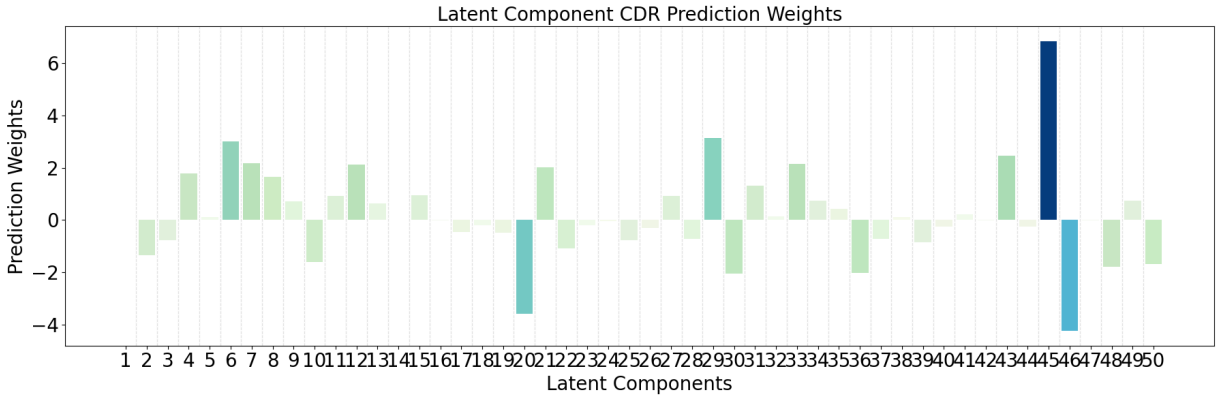
In [5]:
```
# load prediction, modality weights

component_weight_path = root_path + "/prediction_weights.csv"

modality_weight_path = root_path + "/modality_weights.csv"
modality_list = ['CT','TAU', 'PSA', 'GM', 'AMY']
prediction_list = ['CDR'] #["Affective Symptoms", "Hyperactivity"]# ["Psycho

 #["CDR", "Sex", "Age", "GDS", "NPI"]
```

```
# ["Delusion", "Hallucination", "Agitation/Aggression", "Depression/Dysphori
# ["Delusion",  "Agitation/Aggression", "Depression/Dysphoria", "Anxiety",
# ["weight"]
# modality_weight_path_update = root + relative_folder_path + "dd_modality_w
# weight_predict_path = root + relative_folder_path + "prediction_weights.pr
# modalitiy_weight_save_path = root + relative_folder_path + "modality_weigh
# modalitiy_prediction_weight_save_path = root + relative_folder_path + "moc


component_weight_df = pd.read_csv(component_weight_path, header=None, names=
modality_weight_df = pd.read_csv(modality_weight_path, header=None, names=mo
modality_weight_df = modality_weight_df.round(2)
# Reset the index to start from 1
modality_weight_df.index = range(1, len(modality_weight_df) + 1)
```

Step 1. Visualize trained component and modality weights

1.1 Component weights

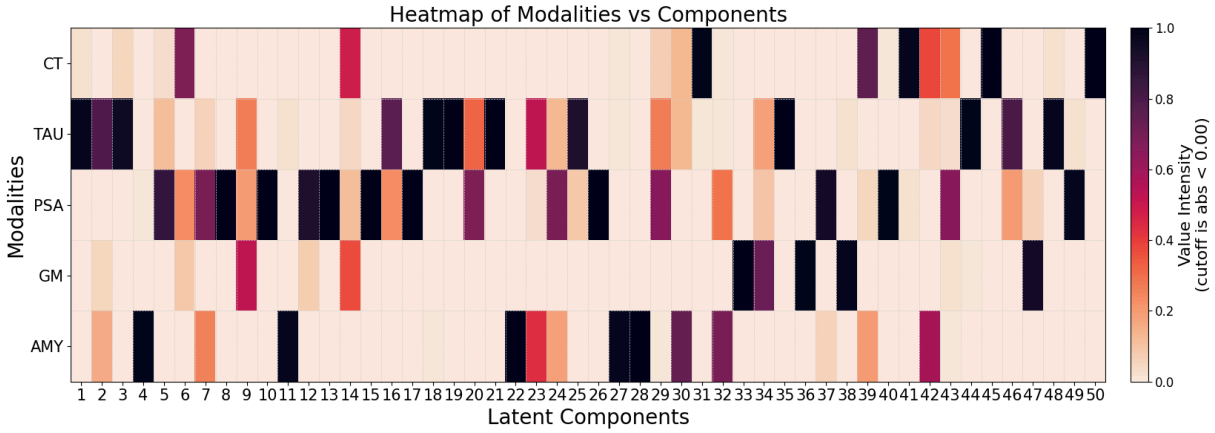
In [12]:
```
# Call the function to plot

for i in range(len(prediction_list)):
    plot_component_weights(
        component_weight_df=component_weight_df,
        title="Latent Component "+ prediction_list[i]+ " Prediction Weights"
        xlabel="Latent Components",
        ylabel="Prediction Weights",
        cmap='GnBu',
        figsize=(18, 6),
        save_path=None,  # Provide a file path to save the plot, e.g., "weig
        colname = prediction_list[i]
    )
```



1.2 Modality weights

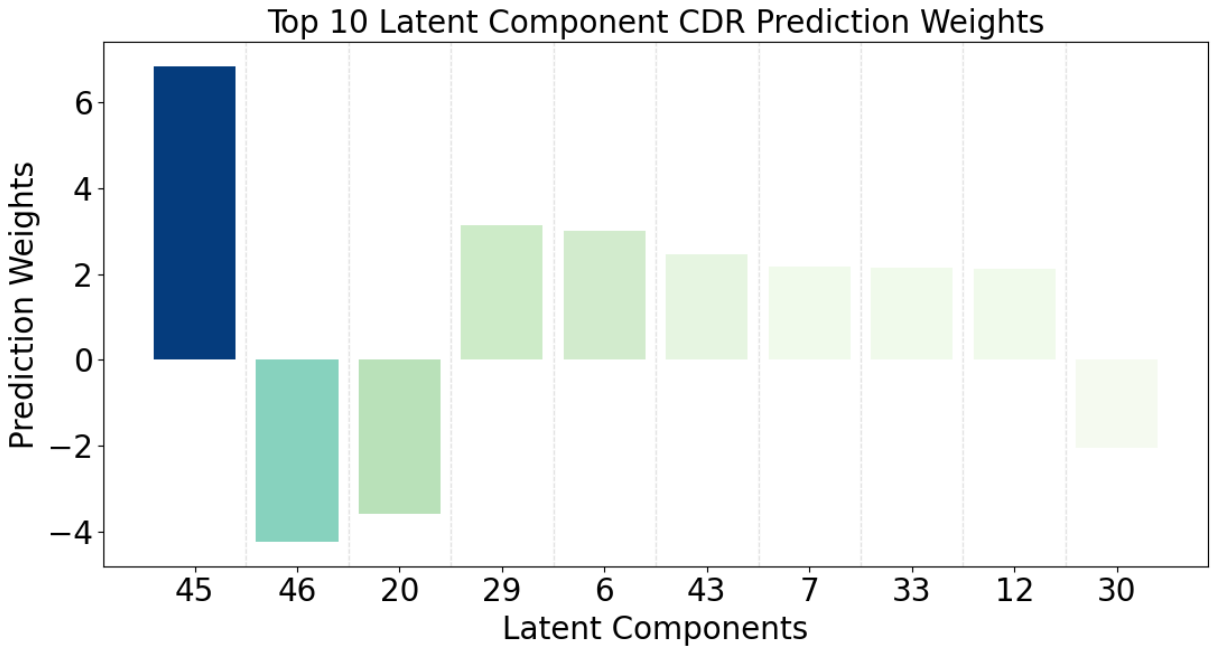
In [7]:
```
# Call the function to plot the heatmap
plot_heatmap_with_labels(
```

```
    data=modality_weight_df,
    row_labels=modality_weight_df.index.tolist(),  # Columns are now rows af
    col_labels=modality_weight_df.columns.tolist(),   # Rows are now columns
    title="Heatmap of Modalities vs Components",
    xlabel="Latent Components",
    ylabel="Modalities",
    cmap="rocket",
    figsize=(18, 6),
    save_path=None,  # Specify a file path to save the figure
    cutoff=0

)
```



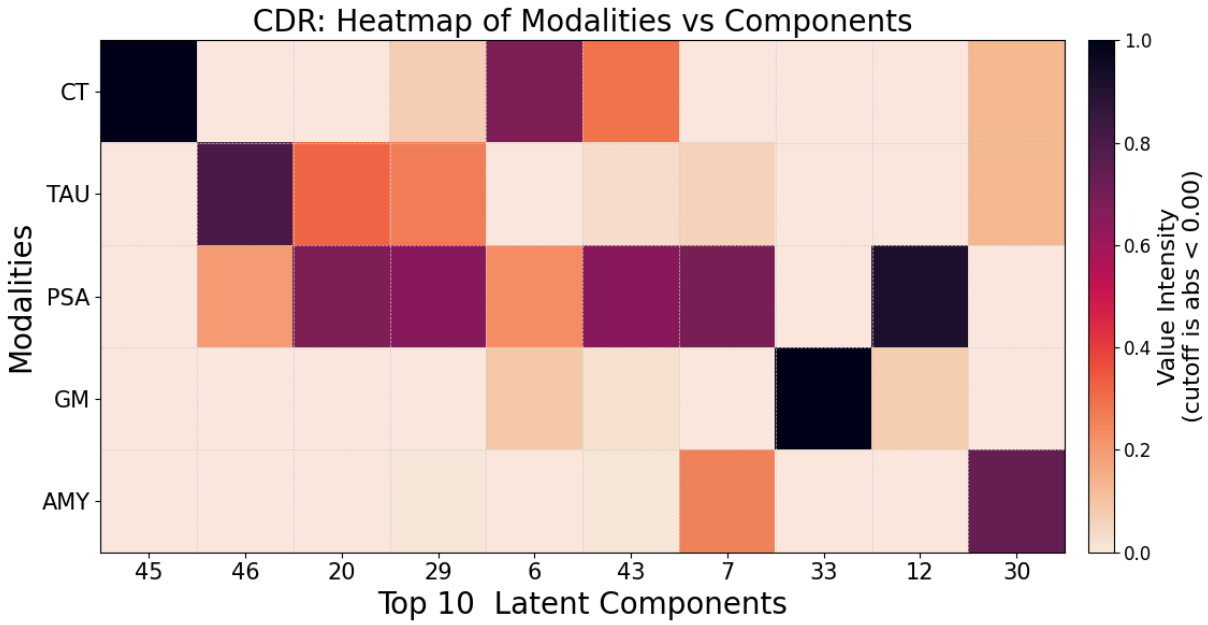
1.3 Top 10 Component weights

In [8]:
```
for i in range(len(prediction_list)):
    component_weight_df_top10 = component_weight_df.sort_values(by=predictic

    plot_component_weights(
    component_weight_df=component_weight_df_top10,
    title="Top 10 Latent Component "+ prediction_list[i]+ " Prediction Weigh
    xlabel="Latent Components",
    ylabel="Prediction Weights",
    cmap='GnBu',
    figsize=(11, 6),
    save_path=None,  # Provide a file path to save the plot, e.g., "weights_
    colname = prediction_list[i]

)
```

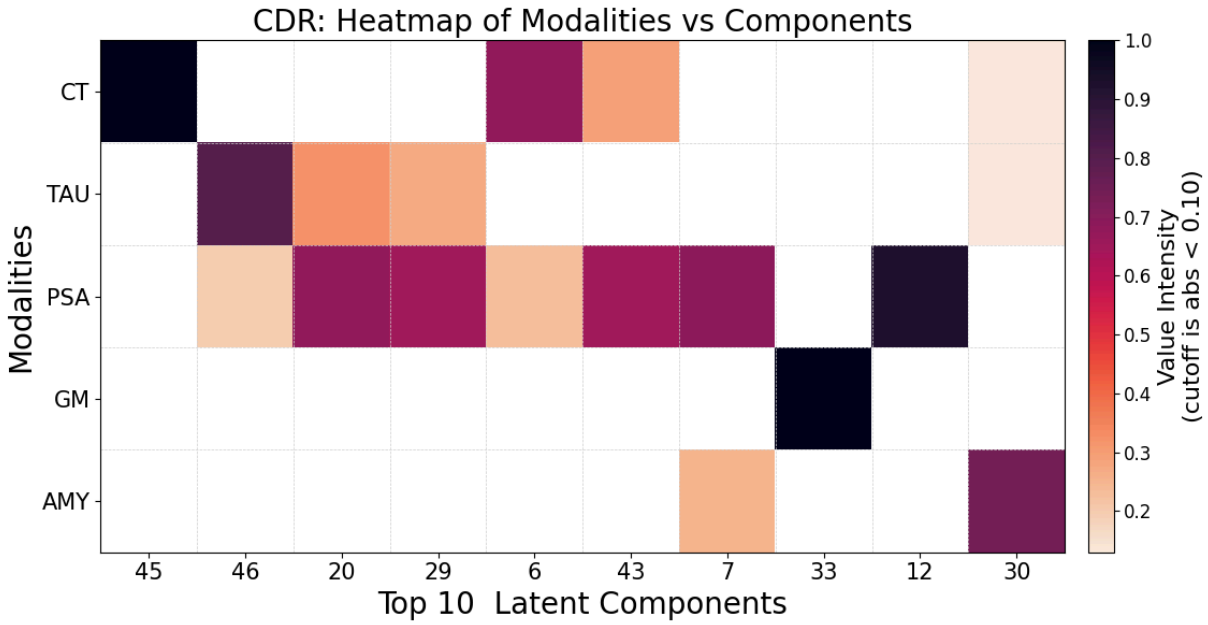


1.4 Modality weights of top 10 components

In [9]:
```
for i in range(len(prediction_list)):
    component_weight_df_top10 = component_weight_df.sort_values(by=predictic

    top10_index = [i+1 for i in component_weight_df_top10.index.to_list()]

    modality_weight_df_top10 = modality_weight_df[modality_weight_df.index.i

    plot_heatmap_with_labels(
    data=modality_weight_df_top10,
    row_labels=modality_weight_df_top10.index.tolist(),  # Columns are now r
    col_labels=modality_weight_df_top10.columns.tolist(),   # Rows are now c
    title= prediction_list[i] + ": Heatmap of Modalities vs Components",
    xlabel="Top 10 " + " Latent Components",
    ylabel="Modalities",
    cmap="rocket",
    figsize=(12, 6),
    save_path=None,  # Specify a file path to save the figure
    cutoff=0
)
```



In [10]:
```
for i in range(len(prediction_list)):
    component_weight_df_top10 = component_weight_df.sort_values(by=predictic

    top10_index = [i+1 for i in component_weight_df_top10.index.to_list()]

    modality_weight_df_top10 = modality_weight_df[modality_weight_df.index.i

    plot_heatmap_with_labels(
    data=modality_weight_df_top10,
    row_labels=modality_weight_df_top10.index.tolist(),  # Columns are now r
    col_labels=modality_weight_df_top10.columns.tolist(),   # Rows are now c
    title= prediction_list[i] + ": Heatmap of Modalities vs Components",
    xlabel="Top 10 " + " Latent Components",
    ylabel="Modalities",
    cmap="rocket",
    figsize=(12, 6),
    save_path=None,  # Specify a file path to save the figure
    cutoff=0.1
)
```


CDR: Heatmap of Modalities vs Components
Modalities
CT
TAU
PSA
GM
AMY
45
46
20
29
6
43
7
33
12
30
Top 10 Latent Components
Value Intensity
(cutoff is abs < 0.10)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2