Introduction

In this report,

- we analyze the miRNA dataset that has been provided in **Machine Learning to Detect Alzheimer's Disease from Circulating Non-coding RNAs** study.

In thier study,

- They computed models to assess the relation between miRNA expression and phenotypes, gender, age, or disease severity (Mini-Mental State Examination; MMSE).
- Of the 21 miRNAs, expression levels of 20 miRNAs were consistently de-regulated in the US and German cohorts. 18 miRNAs were significantly correlated with neurodegeneration.

Report Content

1. Dataset Overview
2. Data Visualization
3. Machine learning

In [1]:
```
# import initial packages
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

1. Dataset Overview

- Cohorts of the patients: United States (US) and Germany (Ger).
- miRNAs Count: 21 known circulating miRNAs.
- number of samples: 465 individuals.
- Biogroups: Alzheimer's disease (AD), mild cognitive impairment (MCI), other neurological diseases (OND), and healthy controls (HC).
- MMSE stands for Mini-Mental State Examination;

In [2]:
```
df = pd.read_excel("./dataset.xls")
```

In [3]:
```
# statistical summary of numeric columns
df.describe().T
```

Out[3]:

	count	mean	std	min	25%	50%	75%	max
Age	464.0	68.614927	11.848111	21.679452	62.000000	70.000000	76.000000	94.600000
MMSE	292.0	23.643836	5.534322	0.000000	20.000000	25.000000	29.000000	30.000000
let-7d-3p	460.0	2.711537	0.823173	-0.280682	2.118270	2.762268	3.327529	4.916473
let-7f-5p	463.0	-1.122261	2.007408	-5.960617	-2.610254	-1.096389	0.113127	5.522197
miR-103a-3p	459.0	-1.074985	1.679834	-4.460554	-2.382302	-1.286426	0.105553	4.929745
miR-107	447.0	5.425687	1.679470	2.041664	4.114993	5.223851	6.512096	11.258179
miR-1285-5p	448.0	6.729128	0.982175	2.423789	6.072618	6.711871	7.370313	10.271582
miR-139-5p	447.0	3.604875	0.994562	-0.133939	3.004365	3.621847	4.191758	8.334491
miR-1468-5p	394.0	7.442343	1.013414	3.176059	6.708120	7.484238	8.113783	11.647386
miR-151-3p	457.0	0.428599	0.959420	-3.950445	-0.193686	0.439392	1.042100	3.639497
miR-17-3p	451.0	4.360017	1.797803	0.487747	3.001158	4.246741	5.697674	9.672257
miR-26a-5p	462.0	-2.786973	1.440377	-6.493584	-3.710726	-3.028545	-2.016259	3.065654
miR-26b-5p	454.0	-1.879367	1.730776	-5.975381	-3.130266	-2.123656	-0.689592	2.921441
miR-28-3p	454.0	3.110536	0.819017	-1.248889	2.571405	3.115088	3.630326	6.117366
miR-3157-3p	302.0	8.685072	0.899224	4.384489	8.155383	8.664401	9.143152	13.379994
miR-345-5p	454.0	3.799081	1.089339	-0.521034	3.024146	3.839405	4.532160	7.313757
miR-34a-5p	288.0	9.174548	1.629890	2.178630	7.991342	8.972309	10.321436	15.372020
miR-361-5p	458.0	4.725931	0.929995	0.248286	4.143349	4.646323	5.274330	8.282001
miR-4482-3p	417.0	7.797488	1.230467	2.213126	7.053278	7.935891	8.669418	11.214571
miR-486-5p	457.0	-9.842494	1.075918	-14.157979	-10.587245	-9.868228	-9.130302	-3.488748
miR-5006-3p	435.0	6.597165	1.000088	1.882285	5.952768	6.643280	7.206893	10.458080
miR-5010-3p	459.0	2.794507	1.133209	-2.012117	2.069377	2.812802	3.569312	7.398596
miR-532-5p	452.0	4.627253	1.518274	0.940303	3.503891	4.649190	5.682517	9.184030

In [4]:
```
# number of samples per each biogroup
```

```
df.Biogroup.value_counts().to_dict()
```

Out[4]:
```
{'HC': 214, 'AD': 145, 'OND': 68, 'MCI': 38}
```

In [5]:
```
# column datatype and number of non-null values per column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 465 entries, 0 to 464
Data columns (total 26 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Cohort       465 non-null    object 
 1   Age          464 non-null    float64
 2   Gender       465 non-null    object 
 3   Biogroup     465 non-null    object 
 4   MMSE         292 non-null    float64
 5   let-7d-3p    460 non-null    float64
 6   let-7f-5p    463 non-null    float64
 7   miR-103a-3p  459 non-null    float64
 8   miR-107      447 non-null    float64
 9   miR-1285-5p  448 non-null    float64
 10  miR-139-5p   447 non-null    float64
 11  miR-1468-5p  394 non-null    float64
 12  miR-151-3p   457 non-null    float64
 13  miR-17-3p    451 non-null    float64
 14  miR-26a-5p   462 non-null    float64
 15  miR-26b-5p   454 non-null    float64
 16  miR-28-3p    454 non-null    float64
 17  miR-3157-3p  302 non-null    float64
 18  miR-345-5p   454 non-null    float64
 19  miR-34a-5p   288 non-null    float64
 20  miR-361-5p   458 non-null    float64
 21  miR-4482-3p  417 non-null    float64
 22  miR-486-5p   457 non-null    float64
 23  miR-5006-3p  435 non-null    float64
 24  miR-5010-3p  459 non-null    float64
 25  miR-532-5p   452 non-null    float64
dtypes: float64(23), object(3)
memory usage: 94.6+ KB
```

Table 1 : Distribution of age, gender, diseases, and MMSE

Metrics. For each of the cohorts and diseases, the number of patients in the US and Germany, the mean and SD for age and MMSE as well as the gender distribution are provided. GER, Germany; MMSE, Mini-Mental State Examination; AD, Alzheimer's disease; OND, other neurological diseases; HC, healthy control; MCI, mild cognitive impairment

---

Result in paper

C

	AD		MCI		HC		OND	
	USA	GER	USA	GER	USA	GER	USA	GER
No of subjects	79	66	17	21	23	191	50	18
Gender (M/F)	40/39	36/30	8/9	10/11	11/12	92/99	16/34	7/11
Age (mean ± SD)	73.2 ± 10.3	72.5 ± 9.3	72.8 ±7.2	70.6 ± 5.3	67.4 ±6.7	69.1 ±8	48.2 ± 13.5	81.7 ± 6.4
MMSE (mean ± SD)	19 ± 3.5	21 ±4.7	25.6 ±1.4	NA	29.4 ± 1.2	28.8 ±1.5	NA	18.8 ± 5.3

In [6]:
```
# our analysis result

biogroups = df.Biogroup.unique().tolist()
cohorts = df.Cohort.unique().tolist()

summary = []
for biogroup in biogroups:
    for cohort in cohorts:
        sub_df = df.loc[(df['Biogroup'] == biogroup) & (df['Cohort'] == cohort)]
        age = "{:.2f} ± {:.2f}".format(sub_df.Age.mean(), sub_df.Age.std())
        mmse= "{:.2f} ± {:.2f}".format(sub_df.MMSE.mean(), sub_df.MMSE.std())
        genders = sub_df['Gender'].value_counts().to_dict()
        count = sub_df.shape[0]
        summary.append([biogroup, cohort, count, age, mmse, str(genders)])

# print the dataframe
pd.DataFrame(summary, columns=["Biogroup","Cohort", "Count", "Age","MMSE","Gender"])
```

Out[6]:

	Biogroup	Cohort	Count	Age	MMSE	Gender
0	AD	GER	66	72.48 ± 9.30	20.98 ± 4.65	{'F': 36, 'M': 30}
1	AD	USA	79	73.19 ± 10.25	18.97 ± 3.54	{'F': 40, 'M': 39}
2	HC	GER	191	69.07 ± 7.95	28.79 ± 1.46	{'M': 99, 'F': 92}
3	HC	USA	23	67.43 ± 6.69	29.39 ± 1.20	{'F': 11, 'M': 10, 'M ': 2}
4	MCI	GER	21	70.62 ± 5.28	nan ± nan	{'M': 11, 'F': 10}
5	MCI	USA	17	72.82 ± 7.19	25.65 ± 1.41	{'M': 9, 'F': 8}
6	OND	GER	18	81.72 ± 6.42	18.78 ± 5.31	{'M': 11, 'F': 7}
7	OND	USA	50	48.12 ± 13.53	nan ± nan	{'M': 34, 'F': 16}

In [ ]:
```
```

2. Data Visualization

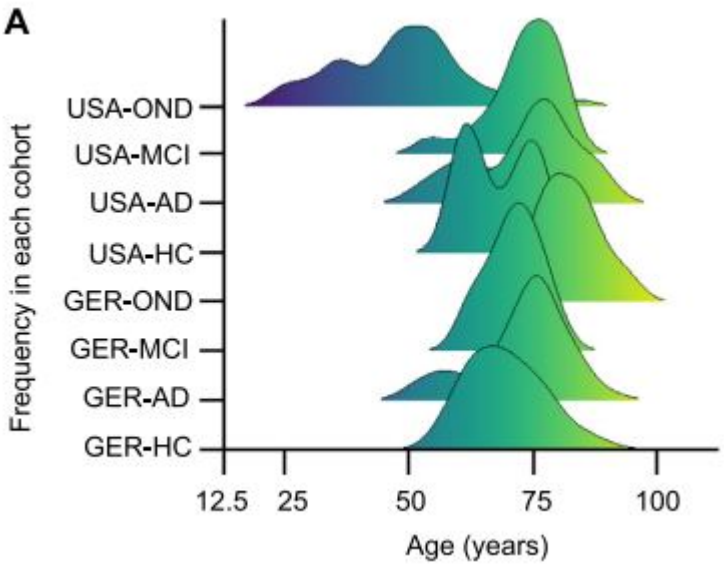
In [7]:
```
df = pd.read_excel("./dataset.xls")
```

In [8]:
```
# make a copy of df
cdf = df.copy()
# new column grouping cohort and biogroup
cdf["cohort-biogroup"] = cdf[['Cohort', 'Biogroup']].agg('-'.join, axis=1)
```

Figure 1.A - Ridge Plot - replication

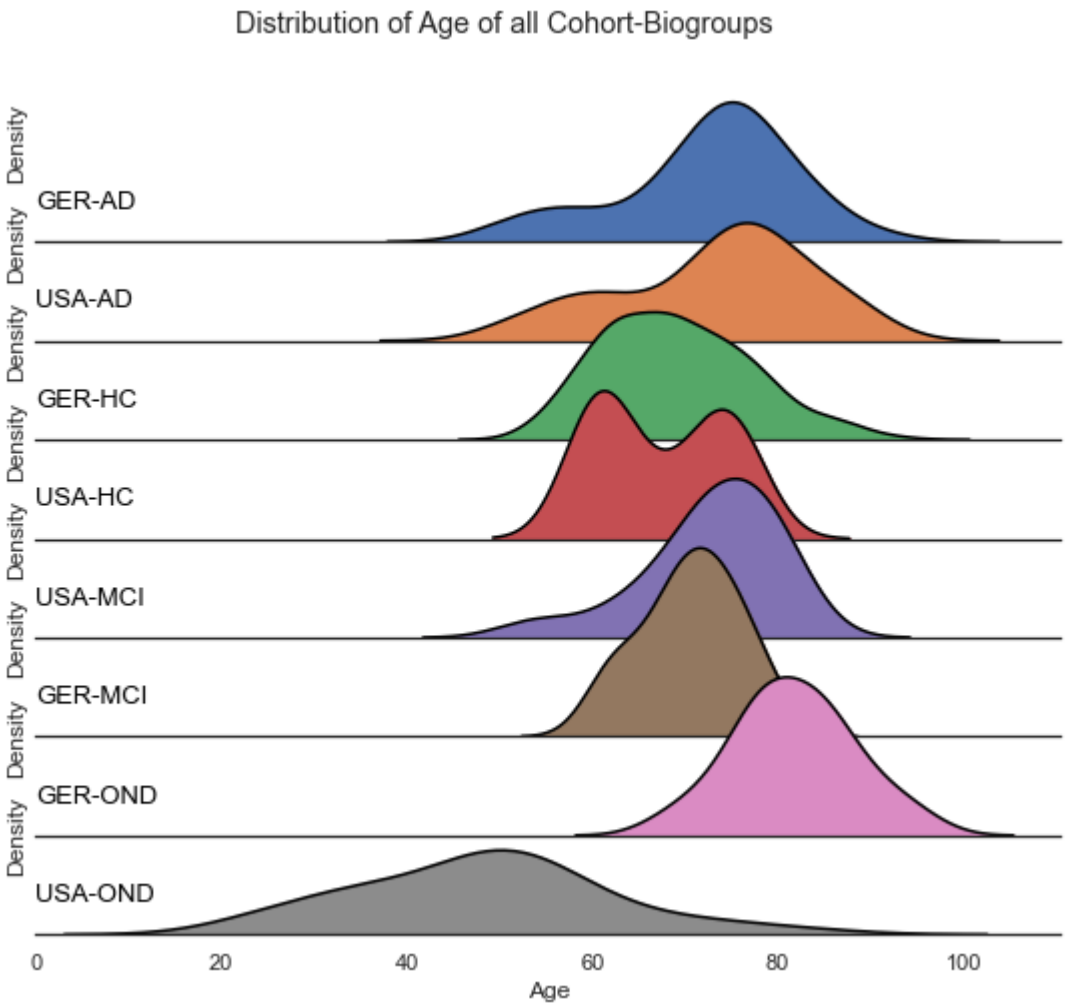
Ridge Plot: Histogram for the age distribution in the different cohorts. The diagram shows for each cohort/disease the age distribution. Only the OND group from the US shows a deviation towards younger patients, while all other groups have similar age ranges.

Result in the paper



In [9]:
```
# replication of the figure

sns.set_theme(style="white", rc={"axes.facecolor": (0, 0, 0, 0)})


g = sns.FacetGrid(cdf, row="cohort-biogroup",hue="cohort-biogroup", aspect=9, height=0
g.map_dataframe(sns.kdeplot, x="Age", fill=True, alpha=1)
g.map_dataframe(sns.kdeplot, x="Age", color='black')

def label(x, color, label):
    ax = plt.gca()
    ax.text(0, .2, label, color='black', fontsize=13,
            ha="left", va="center", transform=ax.transAxes)

g.map(label, "cohort-biogroup")

g.fig.subplots_adjust(hspace=-.5)
g.set_titles("")
g.set(yticks=[], xlabel="Age")
g.despine( left=True)
g.set(xlim=(0, None))
plt.suptitle('Distribution of Age of all Cohort-Biogroups', y=0.98)
```

Out[9]:
```
Text(0.5, 0.98, 'Distribution of Age of all Cohort-Biogroups')
```

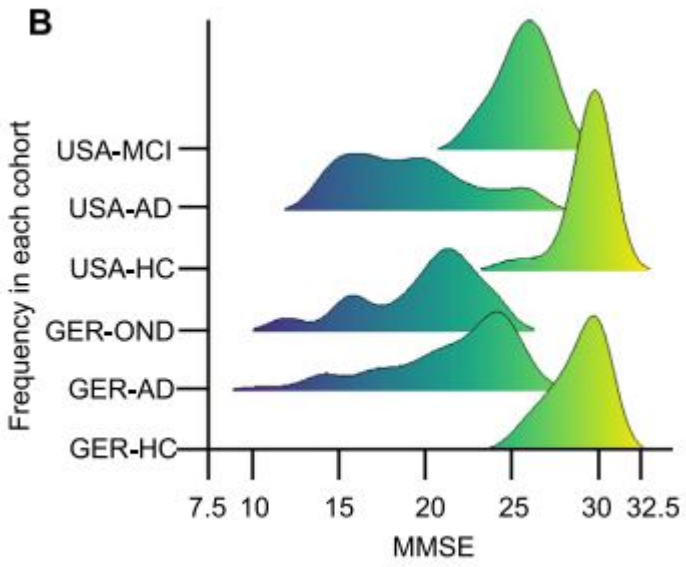


In [ ]:

Figure 1.B - Ridge Plot - replication

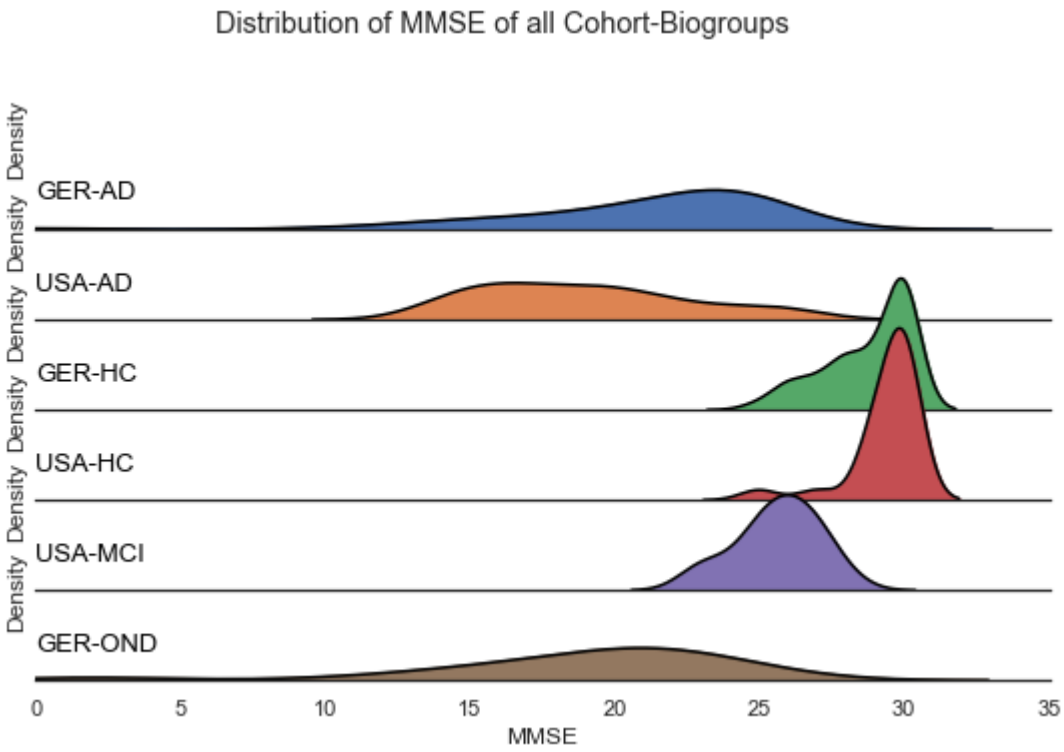
Histogram for the MMSE values. HCs and MCI patients show significantly larger MMSE values as compared to AD and OND patients.

Result in the paper



In [10]:
```
# replicate the figure

# eliminate the empty cohort-biogroup
eliminated = ["GER-MCI", "USA-OND"]
cdf = cdf[~cdf['cohort-biogroup'].isin(eliminated)]

sns.set_theme(style="white", rc={"axes.facecolor": (0, 0, 0, 0)})

g = sns.FacetGrid(cdf,
                  row="cohort-biogroup",
                  hue="cohort-biogroup",
                  aspect=9,
                  height=0.9,
                 )
g.map_dataframe(sns.kdeplot, x="MMSE", fill=True, alpha=1)
g.map_dataframe(sns.kdeplot, x="MMSE", color='black')

def label(x, color, label):
    ax = plt.gca()
    ax.text(0, .2, label, color='black', fontsize=13,
            ha="left", va="center", transform=ax.transAxes)

g.map(label, "cohort-biogroup")

g.fig.subplots_adjust(hspace=-.5)
g.set_titles("")
g.set(yticks=[], xlabel="MMSE")
g.despine( left=True)
g.set(xlim=(0, None))
plt.suptitle('Distribution of MMSE of all Cohort-Biogroups', y=0.98)
```

Out[10]:
```
Text(0.5, 0.98, 'Distribution of MMSE of all Cohort-Biogroups')
```



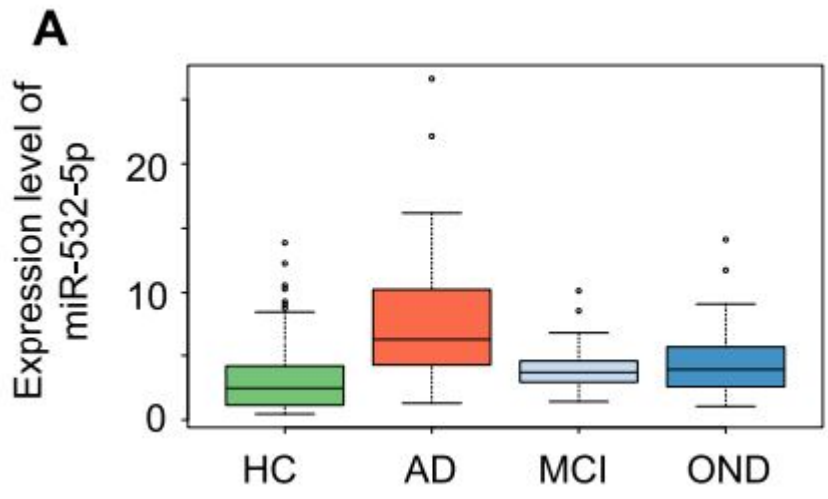
In [ ]:

Figure 2. A - Boxplot - replication

Expression of miR-532-3p. The boxes display the 2nd and 3rd quartile of expression values for miR-532-3p in HC, patients with AD, MCI, or OND. The range of expression values in the four groups is indicated by the error bars with outliers represented by unfilled dots. Median expression of miR-532-3p is indicated as thick black line. miRNAs are specifically dysregulated in the four cohorts and are partially co-expressed

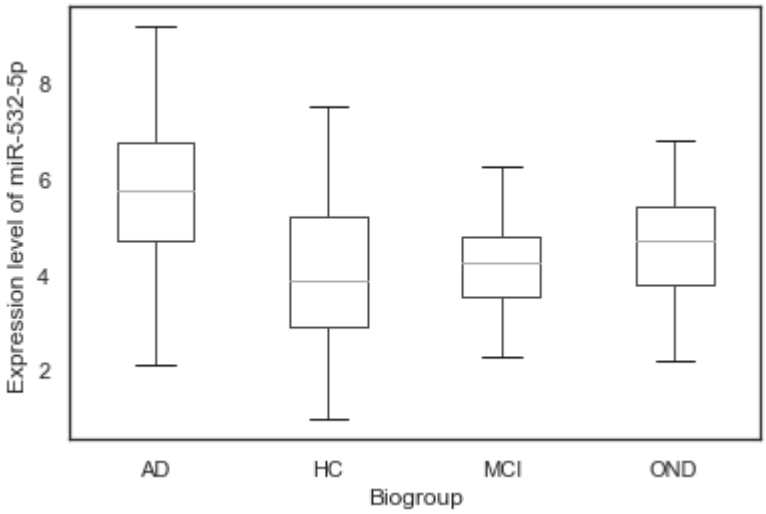
---

Result in the paper



In [11]:
```
mrna = 'miR-532-5p'
bp = df.boxplot(column=mrna, by='Biogroup', grid=False)
bp.set_ylabel(f'Expression level of {mrna}')
bp.set_title('')
#bp.set_ylim(0, 20)
fig = bp.get_figure()
fig.suptitle('')

# result:
# Although their provided data is normalized,
# We can still see >> AD cohort is overexpressed than others
```

Out[11]:
```
Text(0.5, 0.98, '')
```

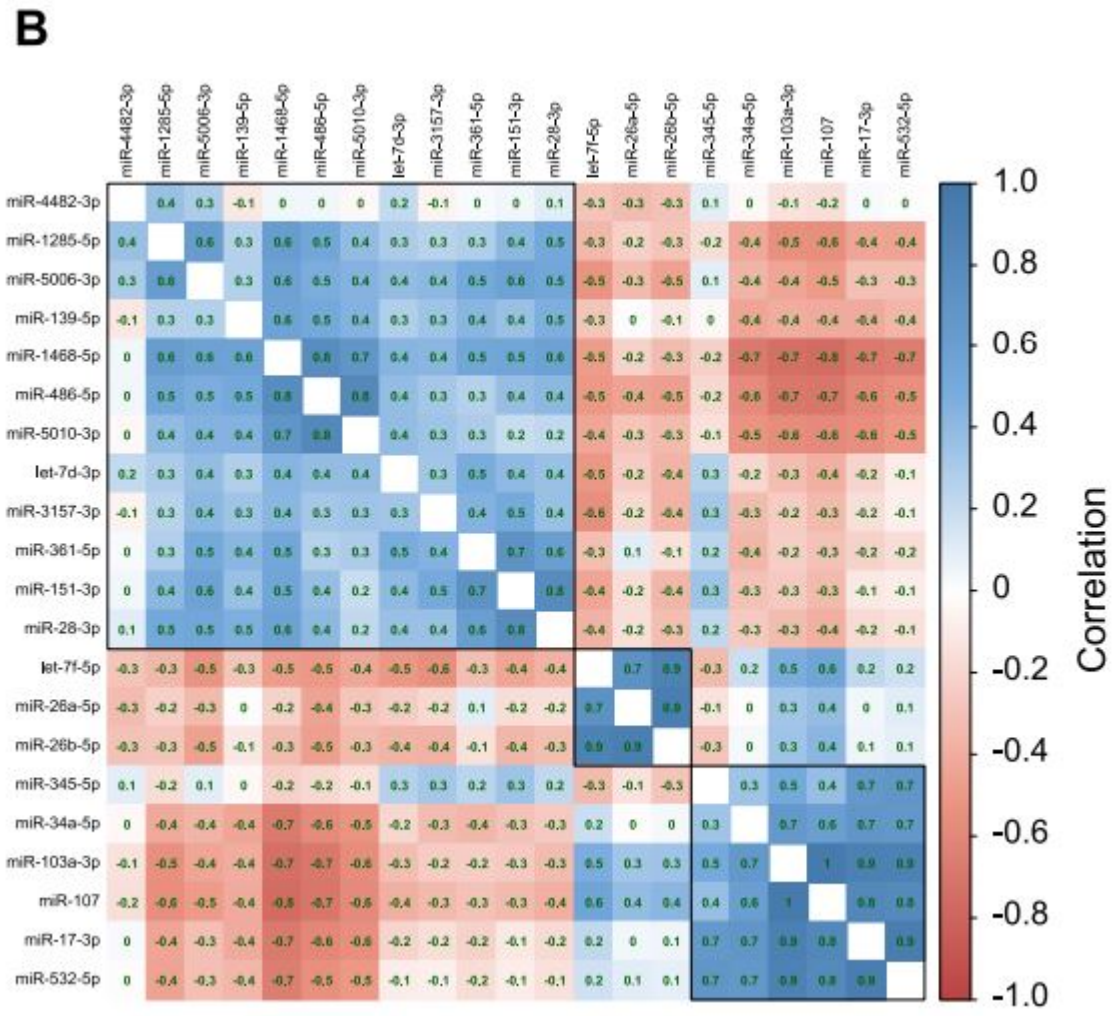


In [ ]:

Figure 2. B - Heatmap - replication

Correlation of miRNA expression. This correlation matrix graphically represents the pair-wise correlation coefficient for all miRNAs tested. According to the color scale on the right side of the matrix, positive and negative correlations are indicated in shades of blue and red, respectively. PCC is given for each pair-wise correlation. Three clusters of miRNAs with highly similar expression patterns are indicated as Clusters A, B, and C on the left side.

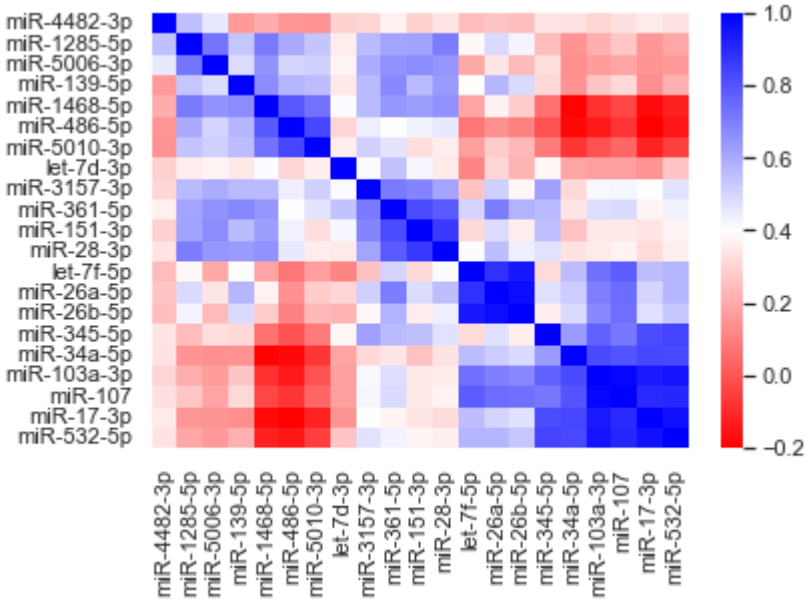
Result of the paper



In [12]:
```
# arrange of columns according to their study heatmap
cols = [
  'miR-4482-3p',
  'miR-1285-5p',
  'miR-5006-3p',
  'miR-139-5p',
  'miR-1468-5p',
  'miR-486-5p',
  'miR-5010-3p',
  'let-7d-3p',
  'miR-3157-3p',
  'miR-361-5p',
  'miR-151-3p',
  'miR-28-3p',
  'let-7f-5p',
  'miR-26a-5p',
  'miR-26b-5p',
  'miR-345-5p',
  'miR-34a-5p',
  'miR-103a-3p',
  'miR-107',
  'miR-17-3p',
  'miR-532-5p',
]
```

```
corr = df.loc[:,'let-7d-3p':].corr().reindex(cols)[cols]

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns, cmap='bwr_r')

# result: the 3 mentioned boxes pattern can be seen
```

Out[12]: <AxesSubplot:>

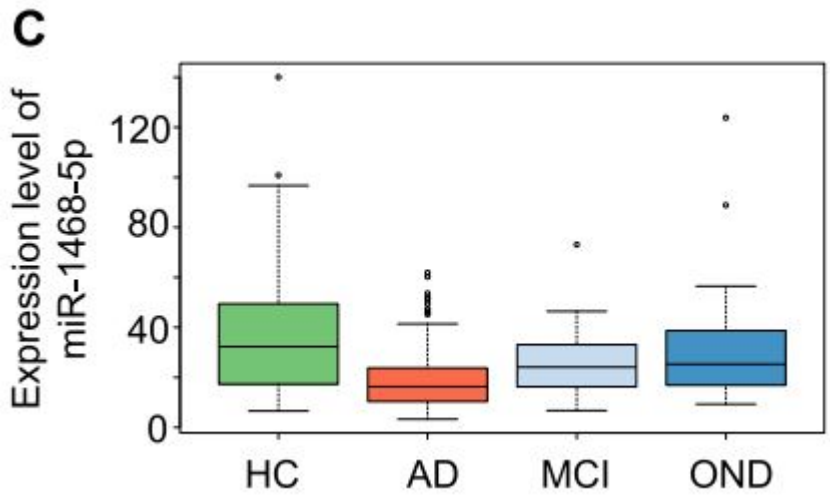


In [ ]:

Figure 2. C - Boxplot

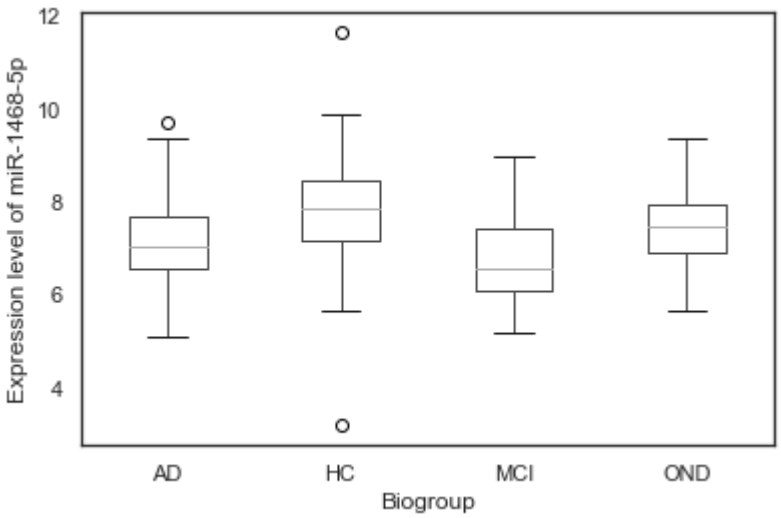
Expression of miR-1468-5p. The boxes display the 2nd and 3rd quartile of expression values for miR-1468-5p in HC, patients with AD, MCI, or OND. The range of expression values in the four groups is indicated by the error bars with outliers represented by unfilled dots. Median expression of miR-1468-5p is indicated as thick black line. PCC, Pearson correlation coefficient.

Result of the paper



In [13]:
```
mrna = 'miR-1468-5p'
bp = df.boxplot(column=mrna, by='Biogroup', grid=False)
bp.set_ylabel(f'Expression level of {mrna}')
bp.set_title('')
#bp.set_ylim(0, 15)
fig = bp.get_figure()
fig.suptitle('')


# result:
# Although their provided data is normalized,
# We can still see >> healthy controls (HC) is overexpressed than others
```

Out[13]:
```
Text(0.5, 0.98, '')
```



3. Machine learning

3.0 LightGbm Model Summary in paper

Model Hyperparms Explanitation

- model performance was assessed using five repetitions of stratified ten-fold cross-validation
- Each repetition was initiated with an integer seed (0–4).
- Thus, in total 50 combinations of different training and validation sets were considered.

- The reported ROC AUC corresponds to the average performance over all repetitions and folds of the model, on data not used for training.

- The models were manually tuned (i.e., no grid search was performed) over the number of leaves (testing ranges between 5 and 50), number of estimators (between 40 and 120), learning rate (0.01 to 0.2), and depth (3 to no restriction).

Their Models Hyperparams

- (CASE #1) The final model comparing patients with AD to all controls uses 30 leaves, a learning rate of 0.1 and 100 estimators.

- (CASE #2) The model comparing patients with AD to unaffected controls uses 9 leaves, a learning rate of 0.05 and 100 estimators. The depth of both models was not restricted.

- Gradient boosted trees outperformed other tree-based methods such as random forests, or classifiers as Support Vector Machines or Neural Networks.

- As an input for the classification task, the expression matrix of the delta Cq values has been used.

Their Models Performance

- (CASE #1): AD patients vs (HC, OND, and MCI combined)
  - the gradient boosted tree model reached an area under the curve (AUC) of 83.5%
- (CASE #2): AD patients vs unaffected controls (HC)
  - the gradient boosted tree model reached an area under the curve (AUC) of 87.6%

In [14]:
```
# import required packages
import io
import math
import random
import statistics
import numpy as np
import pandas as pd
#
import lightgbm as lgb
from sklearn.model_selection import StratifiedKFold
```

3.1. Data preprocess

In [15]:
```
# Dataset Import
ad_df = pd.read_excel('./dataset.xls')
```

In [16]:
```
target_case = 1          # case.1 - (AD vs. (HC, MSI, OND )) | case.2 - (AD vs. HC)
model_params = {}        # initiated for each case independtly
tt_df = ad_df.copy()
```

CASE #1

In [17]:
```
#  (AD vs. (HC, MSI, OND ))

if target_case == 1:

    # split data
    tt_df['class'] = ""
    tt_df.loc[tt_df['Biogroup'] == "AD", 'class'] = 1
    tt_df.loc[tt_df['Biogroup'] != "AD", 'class'] = 0
    tt_df = tt_df.loc[:, 'let-7d-3p':"class"] #

    # set model params (from their study)
    model_params = {
        'num_leaves': 30,
        'learning_rate': 0.1,
        "max_depth": -1,
        #"n_estimators": 100      # default
    }
```

CASE #2

In [18]:
```
# (AD vs. HC)

if target_case == 2:

    # split data
    tt_df['class'] = ""
    tt_df = tt_df[tt_df['Biogroup'].isin(['AD','HC'])]
    tt_df.loc[tt_df['Biogroup'] != "HC", 'class'] = 1
    tt_df.loc[tt_df['Biogroup'] == "HC", 'class'] = 0
    tt_df = tt_df.loc[:, 'let-7d-3p':"class"]

    # set model params (from their study)
    model_params = {
        'num_leaves': 9,
        'learning_rate': 0.05,
        "max_depth": -1,
        #"n_estimators": 100     # default
    }
```

3.2 Model's Performance Evaluation

- Here, we assess the LightGbm binary classifier model using five repetitions of stratified ten-fold cross-validation.
- Together with provided hyperparameters that are provided in the paper; we report the model performance.
- Each repetition was initiated with an integer seed (0–4)

In [19]:
```
# Dataset split
```

```
y = tt_df['class'].astype(int).values
X = tt_df.drop(["class"],axis=1).values
```

In [20]:
```
# LightGBM parameters
params = {
    'task':'train',
    'boosting_type': 'gbdt',
    'is_unbalance':'true',
    'objective': 'binary',
    'metric': 'auc',
    'num_leaves': 9,
    'learning_rate': 0.05,
    "verbose":-1,
    "feature_fraction":0.5,
     **model_params
}
```

In [21]:
```
# Note: to run for another case(1/2): change target_case variable upside
# perform 10 CV of 5 seeded iterations

n_iter = 5
K = 10
val_scores = []

for i in range(n_iter):

    # seed: 0 .. n_iter
    random.seed(i)

    skfold = StratifiedKFold(n_splits=K, shuffle=True)

    print(f"iteration: {i+1}/{n_iter}")
    for fold_id, (train_ids, valid_ids) in enumerate(skfold.split(X, y)):

        lgb_train = lgb.Dataset(X[train_ids, :], y[train_ids])
        lgb_valid  = lgb.Dataset(X[valid_ids, :], y[valid_ids])

        res = {}

        gbm = lgb.train(params,
                        train_set=lgb_train,
                        valid_sets=[lgb_valid],
                        valid_names=['valid'],
                        callbacks=[lgb.record_evaluation(res),
                                   lgb.log_evaluation(100)],
                       )

        val_scores.append(res['valid']['auc'][-1])

    print(" - " * 10)

print(f'Case {target_case}')
print("Average AUC on ValidationSets From %d CV and %d Iterations" % (K, n_iter))
print("Mean = %4f, Std = %4f" % (np.mean(val_scores), np.std(val_scores)))
```

```
iteration: 1/5
[100]   valid's auc: 0.9
[100]   valid's auc: 0.783333
[100]   valid's auc: 0.783333
[100]   valid's auc: 0.84375
[100]   valid's auc: 0.860417
[100]   valid's auc: 0.761161
[100]   valid's auc: 0.890625
[100]   valid's auc: 0.796875
[100]   valid's auc: 0.883929
[100]   valid's auc: 0.790179
 -  -  -  -  -  -  -  -  -  -
iteration: 2/5
[100]   valid's auc: 0.775
[100]   valid's auc: 0.84375
[100]   valid's auc: 0.922917
[100]   valid's auc: 0.816667
[100]   valid's auc: 0.8875
[100]   valid's auc: 0.743304
[100]   valid's auc: 0.736607
[100]   valid's auc: 0.930804
[100]   valid's auc: 0.743304
[100]   valid's auc: 0.868304
 -  -  -  -  -  -  -  -  -  -
iteration: 3/5
[100]   valid's auc: 0.83125
[100]   valid's auc: 0.804167
[100]   valid's auc: 0.933333
[100]   valid's auc: 0.804167
[100]   valid's auc: 0.854167
[100]   valid's auc: 0.912946
[100]   valid's auc: 0.790179
[100]   valid's auc: 0.841518
[100]   valid's auc: 0.814732
[100]   valid's auc: 0.90625
 -  -  -  -  -  -  -  -  -  -
iteration: 4/5
[100]   valid's auc: 0.785417
[100]   valid's auc: 0.89375
[100]   valid's auc: 0.864583
[100]   valid's auc: 0.8
[100]   valid's auc: 0.875
[100]   valid's auc: 0.801339
[100]   valid's auc: 0.863839
[100]   valid's auc: 0.819196
[100]   valid's auc: 0.839286
[100]   valid's auc: 0.767857
 -  -  -  -  -  -  -  -  -  -
iteration: 5/5
[100]   valid's auc: 0.810417
[100]   valid's auc: 0.80625
[100]   valid's auc: 0.927083
[100]   valid's auc: 0.829167
[100]   valid's auc: 0.85
[100]   valid's auc: 0.90625
[100]   valid's auc: 0.933036
[100]   valid's auc: 0.863839
[100]   valid's auc: 0.857143
[100]   valid's auc: 0.779018
 -  -  -  -  -  -  -  -  -  -
```

```
Case 1
Average AUC on ValidationSets From 10 CV and 5 Iterations
Mean = 0.838539, Std = 0.054144
```

Model Results Overview

CASE #1 result sample

- Average AUC on ValidationSets From 10 CV and 5 Iterations
- AUC :: Mean = 0.836768, Std = 0.059347

CASE #2 result sample

- Average AUC on ValidationSets From 10 CV and 5 Iterations
- AUC :: Mean = 0.873266, Std = 0.058003

Similar to what has been reported in their study, we found that the LightGbm achieved a higher average AUC on {Case #2: (AD vs. HC)} than {Case #1 (AD vs. (HC, MSI, OND))}. And the final validation results are almost similar to those have been reported in their study.

In [ ]: