


```
adata_sub.obs['stage'] = adata_sub.obs['stage'].astype('category')
```

```
In [ ]: ##quick scvi run
scvi.model.SCVI.setup_anndata(adata, #anndata object
                               batch_key = 'batch', #categorical key for batch
                               continuous_covariate_keys = ['S_score', 'G2M_score',
                                                            #'total_counts',
                                                            #'pct_counts_mt',
                                                            ], #continuous key
                               layer = 'original_counts') #layer with unmodified
adata_model = scvi.model.SCVI(adata,
                               n_latent = 40,
                               #n_layers = 2,
                               ) #n_latent number of dimensions in latent space
                               #n_layers number of layers
adata_model.view_anndata_setup(adata) #view model configuration
adata_model.train(
    max_epochs = 500
) #change how many runs it should train on
#latent space parameters, layers and epochs affect the final representation.
#make them too high and things will overfit, too low and things will underfit
adata.obsm['X_scvi'] = adata_model.get_latent_representation() #add latent space
#calculate neighbors using scvi latent space instead of a pca
sc.pp.neighbors(adata, #name of anndata object
                use_rep = 'X_scvi')
#clustering
#key added variable is the name of the column that is added
#if empty it will overwrite the 'leiden' column
sc.tl.leiden(adata, #name of anndata object
             resolution = 1)
adata.obsm['X_scvi_MDE'] = scvi.model.utils.mde(adata.obsm['X_scvi'])
```

```
In [ ]: ##score cell cycle
sc.pp.normalize_total(adata, target_sum=1e4)
sc.pp.log1p(adata)
#sc.pp.scale(adata, max_value=10)
#adata = batch_scale(adata)
#cell_cycle_genes = [x.strip() for x in open('regev_lab_cell_cycle_genes.txt')]
cell_cycle_genes = ['Mcm5', 'Pcna', 'Tyms', 'Fen1', 'Mcm2', 'Mcm4', 'Rrm1', 'Ung', '
                    'Cdca7', 'Dtl', 'Prim1', 'Uhrf1', 'Cenpu', 'Hells', 'Rfc2', 'Rfc3',
                    'Rad51ap1', 'Gmnn', 'Wdr76', 'Slbp', 'Ccne2', 'Ubr7', 'Polld3',
                    'Rad51', 'Rrm2', 'Cdc45', 'Cdc6', 'Exo1', 'Tipin', 'Dsccl', 'Blat',
                    'Usp1', 'Clspn', 'Pola1', 'Chaf1b', 'Brip1', 'E2f8', 'Hmgb2', '
                    'Ube2c', 'Birc5', 'Tpx2', 'Top2a', 'Ndc80', 'Cks2', 'Nuf2', 'Ckap2',
                    'Tpo', 'Cenpf', 'Tacc3', 'Pimreg', 'Smc4', 'Ccnb2', 'Ckap2l',
                    'Bub1', 'Kif11', 'Anp32e', 'Tubb4b', 'Gtse1', 'Kif20b', 'Hjurp',
                    'Jpt1', 'Cdc20', 'Ttk', 'Cdc25c', 'Kif2c', 'Rangap1', 'Ncapd2',
                    'Cdca2', 'Cdca8', 'Ect2', 'Kif23', 'Hmnr', 'Aurka', 'Psrc1', 'Aurkb',
                    'Ckap5', 'Cenpe', 'Ctcf', 'Nek2', 'G2e3', 'Gas2l3', 'Cbx5', 'Ctcf']

s_genes = cell_cycle_genes[:43]
g2m_genes = cell_cycle_genes[43:]
cell_cycle_genes = s_genes + g2m_genes
```

```

cell_cycle_genes = [x for x in cell_cycle_genes if x in adata.var_names]
len(cell_cycle_genes)
sc.tl.score_genes_cell_cycle(adata, s_genes=s_genes, g2m_genes=g2m_genes)

```

```

In [ ]: ##add multiple res clustering in a for loop
leiden_res = [0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.6,1.8,2,2.2,2.4,2.6,2.8,3]
res_keys = []
for i in leiden_res:
    sc.tl.leiden(adata, resolution = i, key_added = 'leiden'+str(i))
    res_keys = res_keys + ['leiden'+str(i)]

#plot all resolutions
sc.pl.embedding(
    adata,
    basis='X_scvi_MDE',
    color=res_keys,
    #groups='17',
    frameon=False,
    ncols=3,
    cmap = reds,
    vmin=0.1
)

```

```

In [ ]: #load adata and delete log1p

try:
    del adata.uns['log1p']
except:
    print("no log1p")

```

```

In [ ]: #Interactive dash app to view 2D plot

#create quick dataframe
tmp_df = pd.DataFrame(adata.obsm['X_scvi_MDE'], columns = ['X','Y'])
tmp_df.index = adata.obs.index
tmp_df = tmp_df.join(adata.obs)
tmp_df['barcode'] = list(tmp_df.index)

app = Dash(__name__)

app.layout = html.Div([
    html.Div([
        html.Div([
            html.Label("Select gene"),
            dcc.Dropdown(adata.var.index.tolist(),
                        value='Sox10',
                        id="dropdown_var",
                        ),
            html.Label("Point size"),
            dcc.Slider(0, 5,
                      marks = None,
                      value=1.5,
                      id='slider_marker_size',

```

```

        tooltip={"placement": "bottom", "always_visible": True
    },
    ],
    style={'width': '49%', 'display': 'inline-block'}),
html.Div([
    html.Label("Select obs"),
    dcc.Dropdown(adata.obs.columns.tolist(),
                 value='stage',
                 id="dropdown_obs",
                 ),
    html.Label("Stages"),
    dcc.Checklist(options=[{'label': 'E8', 'value': '8'},
                          {'label': 'E9', 'value': '9'},
                          {'label': 'E10', 'value': '10'},
                          {'label': 'E11', 'value': '11'},
                          {'label': 'E12', 'value': '12'},
                          {'label': 'E13', 'value': '13'},
                          {'label': 'E14', 'value': '14'}],
                 value=['8', '9', '10', '11', '12', '13', '14'],
                 id='checklist_stage',
                 inline=True,
                 ),
    ],
    style={'width': '49%', 'float': 'right', 'display': 'inline-block'
    },
    style={'padding': '10px 5px'}),
html.Div([
    dcc.Graph(id="graph1"),
    html.Label(['Barcode:']),
    html.Pre(id='barcode', children=[]),
    ],
    style={'width': '49%', 'display': 'inline-block'}),
html.Div([
    dcc.Graph(id="graph2")
    ],
    style={'width': '49%', 'display': 'inline-block'})
])

```

```

@app.callback(
    Output("graph1", "figure"),
    Input("dropdown_var", "value"),
    Input("slider_marker_size", "value"),
    Input("checklist_stage", "value"),
)
def update_graph_1(color_var, size_var, stage_var):
    plot_df = tmp_df[tmp_df['stage'].isin(stage_var)]
    fig = px.scatter(plot_df, x='X', y='Y',
                    color = adata[adata.obs['stage'].isin(stage_var)][:,
                    hover_data={'X': False,
                                'Y': False,
                                "stage": True,
                                'barcode': True},
                    color_continuous_scale = plotly_recs,

```

```

        labels = {'color': color_var},
        width=1000, height=1000,
    )
    fig.update_traces(marker={'size': size_var,
                              'line': {'width': .02, 'color': 'DarkSlateGrey'}},
                      })
    fig.layout.scene.camera.projection.type = "orthographic"
    fig.update_layout(uirevision='constant')
    return fig

@app.callback(
    Output("graph2", "figure"),
    Input("checkboxlist_stage", "value"),
    Input("dropdown_obs", "value"),
    Input("slider_marker_size", "value"),
)
def update_graph_1(stage_var, color_var, size_var):
    plot_df = tmp_df[tmp_df['stage'].isin(stage_var)]
    sorted_list = []
    for i in plot_df:
        if pd.api.types.is_categorical_dtype(plot_df[i]):
            plot_df[i] = plot_df[i].cat.remove_unused_categories()
    if pd.api.types.is_categorical_dtype(plot_df[color_var]):
        try:
            sorted_list = sorted(list(plot_df[color_var].unique().categories))
        except:
            sorted_list = sorted(list(plot_df[color_var].unique().categories))

    fig = px.scatter(plot_df, x='X', y='Y',
                     color = plot_df[color_var],
                     hover_data={'X': False,
                                 'Y': False,
                                 "stage": True,
                                 'barcode': True,
                                 'annotation': True,
                                 'annotation_coarse': True},
                     #color='leiden_0.5',
                     color_continuous_scale = plotly_recs,
                     category_orders={color_var: sorted_list},
                     width=1000, height=1000,
                     )

    fig.update_traces(marker={'size': size_var,
                              'line': {'width': .02, 'color': 'DarkSlateGrey'}},
                      })
    fig.layout.scene.camera.projection.type = "orthographic"
    fig.update_layout(legend= {'itemsizing': 'constant'})

    fig.update_layout(uirevision='constant')
    return fig

@app.callback(
    Output('barcode', 'children'),
    Input('graph1', 'clickData'))
def display_click_data(hoverData):

```



```

        {'label': 'E11', 'value': '11'},
        {'label': 'E12', 'value': '12'},
        {'label': 'E13', 'value': '13'},
        {'label': 'E14', 'value': '14'}],
        value=['8', '9', '10', '11', '12', '13', '14'],
        id='checklist_stage',
        inline=True,
    ),
],
    style={'width': '49%', 'float': 'right', 'display': 'inline-block'
},
    style={'padding': '10px 5px'}),
html.Div([
    dcc.Graph(id="graph1"),
    html.Label(['Barcode:']),
    html.Pre(id='barcode', children=[]),
],
    style={'width': '49%', 'display': 'inline-block'}),
html.Div([
    dcc.Graph(id="graph2")
],
    style={'width': '49%', 'display': 'inline-block'})
])

```

```

@app.callback(
    Output("graph1", "figure"),
    Input("dropdown_var", "value"),
    Input("slider_marker_size", "value"),
    Input("checklist_stage", "value"),
)
def update_graph_1(color_var, size_var, stage_var):
    plot_df = tmp_df[tmp_df['stage'].isin(stage_var)]
    fig = px.scatter(plot_df, x='X', y='Y',
                     color = adata[adata.obs['stage'].isin(stage_var)][:,
                     hover_data={'X': False,
                                   'Y': False,
                                   "stage": True,
                                   'barcode': True,
                                   annotation_name: True},
                     color_continuous_scale = plotly_reds,
                     labels = {'color': color_var},
                     width=1000, height=1000,
                     )
    fig.update_traces(marker={'size': size_var,
                              'line': {'width': .02, 'color': 'DarkSlateGrey'}},
                      )
    fig.layout.scene.camera.projection.type = "orthographic"
    fig.update_layout(uirevision='constant')
    return fig

```

```

@app.callback(
    Output("graph2", "figure"),
    Input("checklist_stage", "value"),

```

```

    Input("dropdown_obs", "value"),
    Input("slider_marker_size", "value"),
)
def update_graph_1(stage_var, color_var, size_var):
    plot_df = tmp_df[tmp_df['stage'].isin(stage_var)]
    sorted_list = []
    for i in plot_df:
        if pd.api.types.is_categorical_dtype(plot_df[i]):
            plot_df[i] = plot_df[i].cat.remove_unused_categories()
    if pd.api.types.is_categorical_dtype(plot_df[color_var]):
        try:
            sorted_list = sorted(list(plot_df[color_var].unique().categories))
        except:
            sorted_list = sorted(list(plot_df[color_var].unique().categories))

    fig = px.scatter(plot_df, x='X', y='Y',
                     color = plot_df[color_var],
                     hover_data={'X':False,
                                'Y':False,
                                "stage":True,
                                'barcode':True,
                                annotation_name:True},
                     #color='leiden_0.5',
                     color_continuous_scale = plotly_recs,
                     category_orders={color_var: sorted_list},
                     width=1000, height=1000,
                     )

    fig.update_traces(marker={'size': size_var,
                              'line': {'width':.02, 'color':'DarkSlateGrey'}},
                      )
    fig.layout.scene.camera.projection.type = "orthographic"
    fig.update_layout(legend= {'itemsizing': 'constant'})

    fig.update_layout(uirevision='constant')
    return fig

@app.callback(
    Output('barcode', 'children'),
    Input('graph1', 'clickData'))
def display_click_data(hoverData):
    if hoverData is None:
        return ' '
    else:
        return str(hoverData['points'][0]['customdata'][1])

app.run(jupyter_mode="tab", host='0.0.0.0', port = port , debug=False)

```

In []: *#Interactive dash app to view 3D plot*

```

#create quick dataframe
tmp_df = pd.DataFrame(adata.obsm['X_phate_3D'], columns = ['X','Y','Z'])
tmp_df.index = adata.obs.index
tmp_df = tmp_df.join(adata.obs)
tmp_df['barcode'] = list(tmp_df.index)

```



```

app = Dash(__name__)

app.layout = html.Div([
    html.Div([
        html.Div([
            html.Label("Select gene"),
            dcc.Dropdown(adata.var.index.tolist(),
                        value='Sox10',
                        id="dropdown_var",
                        ),
            html.Label("Point size"),
            dcc.Slider(0, 5,
                      marks = None,
                      value=1.5,
                      id='slider_marker_size',
                      tooltip={"placement": "bottom", "always_visible": True},
                      ),
        ],
        style={'width': '49%', 'display': 'inline-block'}),
    html.Div([
        html.Label("Select obs"),
        dcc.Dropdown(adata.obs.columns.tolist(),
                    value='stage',
                    id="dropdown_obs",
                    ),
        html.Label("Stages"),
        dcc.Checklist(options=[{'label': 'E8', 'value': '8'},
                              {'label': 'E9', 'value': '9'},
                              {'label': 'E10', 'value': '10'},
                              {'label': 'E11', 'value': '11'},
                              {'label': 'E12', 'value': '12'},
                              {'label': 'E13', 'value': '13'},
                              {'label': 'E14', 'value': '14'}],
                    value=['8', '9', '10', '11', '12', '13', '14'],
                    id='checklist_stage',
                    inline=True,
                    ),
    ],
    style={'width': '49%', 'float': 'right', 'display': 'inline-block'}),
    style={'padding': '10px 5px'}),
    html.Div([
        dcc.Graph(id="graph1"),
        html.Label(['Barcode:']),
        html.Pre(id='barcode', children=[]),
    ],
    style={'width': '49%', 'display': 'inline-block'}),
    html.Div([
        dcc.Graph(id="graph2")
    ],
    style={'width': '49%', 'display': 'inline-block'})
])

```

```

@app.callback(
    Output("graph1", "figure"),
    Input("dropdown_var", "value"),
    Input("slider_marker_size", "value"),
    Input("checklist_stage", "value"),
)
def update_graph_1(color_var, size_var, stage_var):
    plot_df = tmp_df[tmp_df['stage'].isin(stage_var)]
    fig = px.scatter_3d(plot_df, x='X', y='Y', z='Z',
                        color = adata[adata.obs['stage'].isin(stage_var)][:,
                        hover_data={'X':False,
                                    'Y':False,
                                    'Z':False,
                                    "stage":True,
                                    'barcode':True},
                        color_continuous_scale = plotly_recs,
                        labels = {'color':color_var},
                        width=1000, height=1000,
                        )
    fig.update_traces(marker={'size': size_var,
                              'line': {'width':.02,'color':'DarkSlateGrey'},
                              })
    fig.layout.scene.camera.projection.type = "orthographic"
    fig.update_layout(uirevision='constant')
    return fig

@app.callback(
    Output("graph2", "figure"),
    Input("checklist_stage", "value"),
    Input("dropdown_obs", "value"),
    Input("slider_marker_size", "value"),
)
def update_graph_1(stage_var, color_var, size_var):
    plot_df = tmp_df[tmp_df['stage'].isin(stage_var)]
    sorted_list = []
    for i in plot_df:
        if pd.api.types.is_categorical_dtype(plot_df[i]):
            plot_df[i] = plot_df[i].cat.remove_unused_categories()
    if pd.api.types.is_categorical_dtype(plot_df[color_var]):
        try:
            sorted_list = sorted(list(plot_df[color_var].unique().categories))
        except:
            sorted_list = sorted(list(plot_df[color_var].unique().categories))

    fig = px.scatter_3d(plot_df, x='X', y='Y', z='Z',
                        color = plot_df[color_var],
                        hover_data={'X':False,
                                    'Y':False,
                                    'Z':False,
                                    "stage":True,
                                    'barcode':True},
                        #color='leiden_0.5',
                        color_continuous_scale = plotly_recs,

```

```

        category_orders={color_var: sorted_list},
        width=1000, height=1000,
    )

    fig.update_traces(marker={'size': size_var,
                              'line': {'width':.02,'color':'DarkSlateGrey'}},
                      })
    fig.layout.scene.camera.projection.type = "orthographic"
    fig.update_layout(legend= {'itemsizing': 'constant'})

    fig.update_layout(uirevision='constant')
    return fig

@app.callback(
    Output('barcode', 'children'),
    Input('graph1', 'clickData'))
def display_click_data(hoverData):
    if hoverData is None:
        return ' '
    else:
        return str(hoverData['points'][0]['customdata'][1])

app.run(jupyter_mode="tab",host='0.0.0.0', port = 11113 ,debug=False)

```

```

In [ ]: from anndata import AnnData
def harmony_integrate_pipe(
    adata: AnnData, #input anndata object
    layer: str = 'original_counts', #layer with unmodified counts
    target_sum: int = 1e4, #target sum for normalization
    max_value: int = 10, #max value for scaling
    batch_key: str = 'batch', #batch key
    rep_key: str = 'X_pca_harmony', #adjusted harmony pca
    mde_key: str = 'X_harmony_MDE', #mde embedding key
    resolution: int = 0.5, #leiden resolution for quick plot
    leiden_key: str = 'leiden_harmony',
) -> AnnData | None:
    if adata.is_view: # we shouldn't need this here...
        adata._init_as_actual(adata.copy())

    adata.X = adata.layers[layer].copy()
    try:
        del adata.uns['log1p']
    except:
        print('no log1p')

    sc.pp.normalize_total(adata, target_sum=target_sum)
    sc.pp.log1p(adata)
    sc.pp.scale(adata, max_value=max_value)
    sc.tl.pca(adata)
    sc.external.pp.harmony_integrate(adata, key = batch_key, adjusted_basis
    sc.pp.neighbors(adata, use_rep = rep_key)
    adata.obsm[mde_key] = scvi.model.utils.mde(adata.obsm[rep_key])
    sc.tl.leiden(adata, resolution = resolution, key_added = leiden_key)
    sc.pl.embedding(adata,
                    basis=mde_key,

```

```
color = [leiden_key, batch_key, 'phase', ],
ncols=3, cmap=reds, vmin=0.3)
```

```
In [ ]: ##3d velocity arrows
import plotly.graph_objects as go
import pandas as pd

fig = go.Figure(data = go.Cone(
    x=umap[:, 0],
    y=umap[:, 1],
    z=umap[:, 2],
    u=velocity[:, 0],
    v=velocity[:, 1],
    w=velocity[:, 2],
    #colorscale='Blues',
    sizemode="scaled",
    sizeref=10,))

fig.update_layout(width=1000, height=1000, scene=dict(aspectratio=dict(x=1, y
    camera_eye=dict(x=1.2, y=1.2, z=0.6)))
fig.layout.scene.camera.projection.type = "orthographic"

fig.show()
```

```
In [ ]: # fig = px.scatter_3d(tmp_df, x='X', y='Y', z='Z',
#                               #color = adata[:, 'Tbx22'].X.flatten(),
#                               color='leiden_0.5',
#                               color_continuous_scale = plotly_reds,
#                               width=1000, height=1000,
#                               )
# fig.update_traces(marker={'size': 1.5,
#                             'line': {'width': .02, 'color': 'DarkSlateGrey'}}
# )
# fig.layout.scene.camera.projection.type = "orthographic"

# fig.show()

# fig = px.scatter_3d(tmp_df, x='X', y='Y', z='Z',
#                       color = adata[:, 'Meox2'].X.flatten(),
#                       #color='leiden_0.5',
#                       color_continuous_scale = plotly_reds,
#                       width=1000, height=1000,
#                       )
# fig.update_traces(marker={'size': 1.5,
#                             'line': {'width': .02, 'color': 'DarkSlateGrey'}},
# )
# fig.layout.scene.camera.projection.type = "orthographic"

# fig.show()
```

```
In [ ]: from anndata import AnnData
def scvi_anndata_test(
```

```

adata: AnnData,
batch_key: str = 'batch',
continuous_covariate_keys: list = ['S_score', 'G2M_score'],
categorical_covariate_keys: list = None,
layer: str='original_counts',
n_latent: int = 20,
n_layers: int = 1,
max_epochs: int = 500,
n_neighbors: int = 15,
resolution: int = 1,
key_added: str | None = None,
leiden_key: str = 'leiden_scvi',
) -> AnnData | None:
    if adata.is_view: # we shouldn't need this here...
        adata._init_as_actual(adata.copy())

    if key_added is None:
        latent_key = "X_scvi"
        mde_key = "X_scvi_MDE"
    else:
        latent_key = key_added + "_scvi"
        mde_key = key_added + "_scvi_MDE"

    scvi.model.SCVI.setup_anndata(adata, #anndata object
                                batch_key = batch_key, #categorical key for
                                continuous_covariate_keys = continuous_covariate_keys,
                                #'total_counts'
                                #'pct_counts'
                                #continuous
                                categorical_covariate_keys = categorical_covariate_keys,
                                layer = layer) #layer with unmodified counts

    adata_model = scvi.model.SCVI(adata,
                                n_latent = n_latent, #n_latent number of components
                                n_layers = n_layers, #n_layers number of layers
                                )

    print(adata_model.view_anndata_setup(adata))

    adata_model.train(
        max_epochs = max_epochs, #change how many runs it should train on
    )

    adata.obsm[latent_key] = adata_model.get_latent_representation()

    sc.pp.neighbors(adata, #name of anndata object
                   n_neighbors = n_neighbors,
                   use_rep = latent_key,
                   )
    sc.tl.leiden(adata, #name of anndata object
                resolution = resolution,
                key_added = leiden_key
                )

    adata.obsm[mde_key] = scvi.model.utils.mde(adata.obsm[latent_key])

```

```

sc.pl.embedding(adata,
                 basis=mde_key,
                 color = [leiden_key, batch_key, 'phase', ],
                 ncols=3, cmap=reds, vmin=0.3)

```

```

In [ ]: from typing import TYPE_CHECKING, Literal, get_args
        _Method = Literal["logreg", "t-test", "wilcoxon", "t-test_overestim_var"]
        def plot_deg_embedding(
            adata: AnnData,
            method: _Method,
            groupby: str = 'leiden',
            key_added: str | None = None,
            cluster_name: str = '0',
            n_genes: int = 100,
            basis: str = 'X_scvi_MDE',
        ) -> AnnData | None:
            avail_methods = set(get_args(_Method))
            if method not in avail_methods:
                raise ValueError(f"Method must be one of {avail_methods}.")

            if key_added is None:
                key_added = "rank_genes_groups_"+method

            sc.set_figure_params(dpi=50, figsize=[6,4]) #small figures otherwise your
            sc.tl.rank_genes_groups(adata, groupby = groupby, method=method, key_added=key_added)
            genes = adata.uns[key_added]['names'][cluster_name][:n_genes].tolist()
            sc.pl.embedding(adata,
                           basis=basis,
                           color=[groupby]+genes,
                           frameon=False,
                           groups=cluster_name,
                           ncols=4,
                           cmap = reds,
                           vmin=0.1,
                           )

```

```

In [ ]: def default_processing_to_mde(
    adata: AnnData, #input anndata object
    layer: str = 'original_counts', #layer with unmodified counts
    target_sum: int = 1e4, #target sum for normalization
    max_value: int = 10, #max value for scaling
    n_neighbors: int = 15,
    n_pcs: int = 30,
    mde_key: str = 'X_MDE', #mde embedding key
    resolution: int = 0.5, #leiden resolution for quick plot
    leiden_key: str = 'leiden_default',
) -> AnnData | None:
    if adata.is_view: # we shouldn't need this here...
        adata._init_as_actual(adata.copy())

    adata.X = adata.layers[layer].copy()
    try:
        del adata.uns['log1p']
    except:
        print('no log1p')

```

```

sc.pp.normalize_total(adata, target_sum=target_sum)
sc.pp.log1p(adata)
sc.pp.scale(adata, max_value=max_value)
sc.tl.pca(adata)
sc.pp.neighbors(adata, n_neighbors = n_neighbors, n_pcs = n_pcs)
adata.obsm[mde_key] = scvi.model.utils.mde(adata.obsm['X_pca'])
sc.tl.leiden(adata, resolution = resolution, key_added = leiden_key)
sc.pl.embedding(adata,
                basis=mde_key,
                color = [leiden_key, 'phase', ],
                ncols=3, cmap=reds, vmin=0.3)

```

```

In [ ]: def score_cell_cycle_mouse(
        adata: AnnData,
    ) -> AnnData | None:
    cell_cycle_genes = ['Mcm5', 'Pcna', 'Tyms', 'Fen1', 'Mcm2', 'Mcm4', 'Rrm1', 'Ubr7', 'Cdca7', 'Dtl', 'Prim1', 'Uhrf1', 'Cenpu', 'Hells', 'Rfc2', 'Rfc1', 'Rad51ap1', 'Gmn', 'Wdr76', 'Slbp', 'Ccne2', 'Ubr7', 'Pold3', 'Rad51', 'Rrm2', 'Cdc45', 'Cdc6', 'Exo1', 'Tipin', 'Dsccl', 'Bl', 'Usp1', 'Clspn', 'Pola1', 'Chaf1b', 'Brip1', 'E2f8', 'Hmgb2', 'Ube2c', 'Birc5', 'Tpx2', 'Top2a', 'Ndc80', 'Cks2', 'Nuf2', 'Ck', 'Tpo', 'Cenpf', 'Tacc3', 'Pimreg', 'Smc4', 'Ccnb2', 'Ckap2l', 'Bub1', 'Kif11', 'Anp32e', 'Tubb4b', 'Gtse1', 'Kif20b', 'Hjrp', 'Jpt1', 'Cdc20', 'Ttk', 'Cdc25c', 'Kif2c', 'Rangap1', 'Ncapd2', 'Cdca2', 'Cdca8', 'Ect2', 'Kif23', 'Hmrr', 'Aurka', 'Psrc1', 'A', 'Ckap5', 'Cenpe', 'Ctcf', 'Nek2', 'G2e3', 'Gas2l3', 'Cbx5', 'Ce']

    s_genes = cell_cycle_genes[:43]
    g2m_genes = cell_cycle_genes[43:]
    cell_cycle_genes = s_genes + g2m_genes
    cell_cycle_genes = [x for x in cell_cycle_genes if x in adata.var_names]
    len(cell_cycle_genes)
    sc.tl.score_genes_cell_cycle(adata, s_genes=s_genes, g2m_genes=g2m_genes)

```

```

In [ ]: def default_regress_cycle_to_mde(
        adata: AnnData, #input anndata object
        layer: str = 'original_counts', #layer with unmodified counts
        target_sum: int = 1e4, #target sum for normalization
        max_value: int = 10, #max value for scaling
        use_hvg_genes: bool = True,
        n_top_genes: int = 3000, #number of top genes
        n_comps: int = 50, #comps of pca to calculate
        n_neighbors: int = 15,
        mde_key: str = 'X_MDE', #mde embedding key
        resolution: int = 0.5, #leiden resolution for quick plot
        leiden_key: str = 'leiden_default',
        skip_preprocess: bool = False
    ) -> AnnData | None:
    if adata.is_view: # we shouldn't need this here...
        adata._init_as_actual(adata.copy())
    if skip_preprocess is False:
        adata.X = adata.layers[layer].copy()
    try:
        del adata.uns['log1p']

```

```

except:
    print('no loglp')

    sc.pp.normalize_total(adata, target_sum=target_sum)
    sc.pp.loglp(adata)
    sc.pp.scale(adata, max_value=max_value)
    if not 'phase' in adata.obs:
        print('cell cycle not scored, scoring now')
        score_cell_cycle_mouse(adata)
    print(str(time.time())+' : Regressing cell cycle')
    sc.pp.regress_out(adata, ['S_score', 'G2M_score'])
    print(str(time.time())+' : Highly Variable Genes')
    if use_hvg_genes is True:
        sc.pp.highly_variable_genes(adata, layer = layer, n_top_genes = n_top_genes)

    print(str(time.time())+' : PCA')
    sc.tl.pca(adata, n_comps=n_comps)
    print(str(time.time())+' : Neighbors')
    sc.pp.neighbors(adata, n_neighbors = n_neighbors)
    print(str(time.time())+' : MDE and UMAP')
    sc.tl.umap(adata),
    adata.obsm[mde_key] = scvi.model.utils.mde(adata.obsm['X_pca'])
    print(str(time.time())+' : Leiden clustering')
    sc.tl.leiden(adata, resolution = resolution, key_added = leiden_key)
    sc.pl.embedding(adata,
                    basis=mde_key,
                    color = [leiden_key, 'phase', ],
                    ncols=3, cmap=reds, vmin=0.3)
    sc.pl.embedding(adata,
                    basis='X_umap',
                    color = [leiden_key, 'phase', ],
                    ncols=3, cmap=reds, vmin=0.3)

```

```

In [ ]: from anndata import AnnData
def harmony_regress_integrate_pipe(
    adata: AnnData, #input anndata object
    layer: str = 'original_counts', #layer with unmodified counts
    target_sum: int = 1e4, #target sum for normalization
    max_value: int = 10, #max value for scaling
    n_comps: int = 50, #comps of pca to calculate
    batch_key: str = 'batch', #batch key
    max_iter_harmony: int = 10,
    random_state: int = 999,
    n_neighbors: int = 15,
    rep_key: str = 'X_pca_harmony', #adjusted harmony pca
    mde_key: str = 'X_harmony_MDE', #mde embedding key
    resolution: int = 0.5, #leiden resolution for quick plot
    leiden_key: str = 'leiden_harmony',
    skip_preprocess: bool = False,
    use_highly_variable: bool = True,
    n_top_genes: int | None = None,
) -> AnnData | None:
    if adata.is_view: # we shouldn't need this here...
        adata._init_as_actual(adata.copy())

    if skip_preprocess is False:

```



```

adata.X = adata.layers[layer].copy()
try:
    del adata.uns['log1p']
except:
    print('no log1p')

sc.pp.normalize_total(adata, target_sum=target_sum)
sc.pp.log1p(adata)
sc.pp.scale(adata, max_value=max_value)
if not 'phase' in adata.obs:
    print('cell cycle not scored, scoring now')
    score_cell_cycle_mouse(adata)
    print(str(time.time())+': Regressing cell cycle')
    sc.pp.regress_out(adata, ['S_score', 'G2M_score'])
print(str(time.time())+': PCA')
if use_highly_variable is True:
    sc.pp.highly_variable_genes(adata, n_top_genes = n_top_genes, batch_
sc.tl.pca(adata,n_comps = n_comps,use_highly_variable = use_highly_varia
sc.external.pp.harmony_integrate(adata,key = batch_key, adjusted_basis
sc.pp.neighbors(adata, use_rep = rep_key, n_neighbors = n_neighbors,)
adata.obsm[mde_key] = scvi.model.utils.mde(adata.obsm[rep_key])
sc.tl.umap(adata)
sc.tl.leiden(adata,resolution = resolution,key_added = leiden_key)
sc.pl.embedding(adata,
                basis=mde_key,
                color = [leiden_key,batch_key,'phase',],
                ncols=3, cmap=reds, vmin=0.3)
sc.pl.embedding(adata,
                basis='X_umap',
                color = [leiden_key,batch_key,'phase',],
                ncols=3, cmap=reds, vmin=0.3)

```

```

In [ ]: def subset_by_cluster_and_gene(
        adata: AnnData,
        gene_name: str = 'Dlx5',
        layer: str = 'original_counts',
        expression_cutoff: int = 2,
        groupby: str = 'leiden',
        group: str = '0',
    ) -> AnnData | None:
    expression_index = adata[(adata[:,gene_name].layers[layer]>expression_cu
    cluster_index = adata[(adata.obs[groupby]==group), : ].obs.index
    exclusion_index = cluster_index[cluster_index.isin(expression_index)]
    inclusion_index = adata.obs.index[~adata.obs.index.isin(exclusion_index)]
    adata = adata[inclusion_index,:].copy()
    return(adata)

```

```

In [ ]: #renaming clusters
adata.obs['annotation'] = adata.obs['leiden'] #create new column
adata.obs['annotation'] = adata.obs['annotation'].cat.add_categories([ ])#ac

adata.obs.loc[adata[adata.obs['leiden'] == '0'].obs.index,'annotation'] = 'r
adata.obs['annotation'] = adata.obs['annotation'].cat.remove_unused_categori

```