Supplemental data for

Modarresi, M. (2026) “Characterization of Iranian rice genetic resources for key grain quality traits”, *Genetic Resources*, 7(13), pp. 153–168. doi: 10.46265/genresj.ROKV2181

Supplemental File 1. Python Script for Path Analysis Diagram of Cooked Grain Length as a Key Trait.

```
import pandas as pd

import numpy as np

import semopy

import matplotlib.pyplot as plt

import networkx as nx

from matplotlib.backends.backend_pdf import PdfPages

from io import StringIO


# Embed the data directly



--------Enter your Data Here---------



# Read data from string

data = pd.read_csv(StringIO(data_str))


# Select numerical variables for path analysis

path_data = data[['Amylose_content', 'Gelatinization_temperature', 'Grain_length',
```

'Grain_width', 'Cooked_grain_length', 'Grain_shape', 'Grain_elongation']]

```
# Define the path model
model = """
# Direct effects
Grain_length ~ Amylose_content + Gelatinization_temperature
Grain_width ~ Amylose_content + Gelatinization_temperature
Grain_shape ~ Grain_length + Grain_width
Cooked_grain_length ~ Grain_length + Grain_width
Grain_elongation ~ Cooked_grain_length + Grain_shape

# Covariances
Amylose_content ~~ Gelatinization_temperature
Grain_length ~~ Grain_width
Grain_shape ~~ Cooked_grain_length
"""

# Fit the path model
mod = semopy.Model(model)
fit = mod.fit(path_data)

# Extract results
results = mod.inspect(std_est=True)  # Use Model.inspect with std_est=True for standardized estimates

# Filter significant paths (p < 0.05)
sig_paths = results[results['p-value'] < 0.05][['lval', 'op', 'rval', 'Est. Std', 'p-value']]
```

```
sig_paths = sig_paths[sig_paths['op'].isin(['~', '~~'])]  # Direct effects and covariances


# Calculate R-squared values manually
r_squared = {}
for var in ['Grain_length', 'Grain_width', 'Grain_shape', 'Cooked_grain_length',
'Grain_elongation']:
    try:
        r2 = mod.inspect(mode='r2').loc[mod.inspect(mode='r2')['Variable'] == var, 'R2'].values
        r_squared[var] = r2[0] if len(r2) > 0 else 0.0
    except:
        r_squared[var] = 0.0  # Default to 0 if R² is not available


# Create edges for significant paths and covariances
edges = []
for _, row in sig_paths.iterrows():
    if row['op'] == '~':
        edges.append((row['rval'], row['lval'], row['Est. Std']))
    elif row['op'] == '~~':
        edges.append((row['lval'], row['rval'], row['Est. Std']))


# Create a directed graph
G = nx.DiGraph()


# Add nodes
nodes = ['Amylose_content', 'Gelatinization_temperature', 'Grain_length',
         'Grain_width', 'Grain_shape', 'Cooked_grain_length', 'Grain_elongation']
G.add_nodes_from(nodes)
```

```
# Add edges with weights (standardized coefficients)
for src, dst, weight in edges:
    G.add_edge(src, dst, weight=weight)

# Define positions for nodes (tree-like layout)
pos = {
    'Amylose_content': (0, 2),
    'Gelatinization_temperature': (2, 2),
    'Grain_length': (0, 1),
    'Grain_width': (2, 1),
    'Grain_shape': (1, 0.5),
    'Cooked_grain_length': (1, 0),
    'Grain_elongation': (1, -1)
}

# Set up figure with 3 cm margins
dpi = 150
width_px, height_px = 1200, 1000
cm_to_inch = 0.393701
margin_cm = 3
margin_in = margin_cm * cm_to_inch
width_in = width_px / dpi
height_in = height_px / dpi
total_width_in = width_in + 2 * margin_in
total_height_in = height_in + 2 * margin_in
```

```
fig, ax = plt.subplots(figsize=(total_width_in, total_height_in))
plt.subplots_adjust(left=margin_in/total_width_in, right=1-margin_in/total_width_in,
                    bottom=margin_in/total_height_in, top=1-margin_in/total_height_in)

# Draw nodes with R-squared values
for node in G.nodes:
    r2 = r_squared.get(node, 0.0)
    label = f"{node}\nR²={r2:.3f}" if r2 > 0 else node
    nx.draw_networkx_nodes(G, pos, nodelist=[node], node_size=3000, node_color="lightblue",
                           node_shape="s", ax=ax)
    nx.draw_networkx_labels(G, pos, labels={node: label}, font_size=8, ax=ax)

# Draw edges with labels (standardized coefficients)
edge_labels = {(u, v): f"{d['weight']:.3f}" for u, v, d in G.edges(data=True)}
for (u, v), label in edge_labels.items():
    weight = G[u][v]["weight"]
    color = "green" if weight > 0 else "red"
    # Draw curved dashed arrows for covariances
    if (u, v) in [('Amylose_content', 'Gelatinization_temperature'),
                  ('Gelatinization_temperature', 'Amylose_content'),
                  ('Grain_length', 'Grain_width'),
                  ('Grain_width', 'Grain_length'),
                  ('Grain_shape', 'Cooked_grain_length'),
                  ('Cooked_grain_length', 'Grain_shape')]:
        rad = 0.3
        nx.draw_networkx_edges(G, pos, edgelist=[(u, v)], arrowstyle="-", style="dashed",
                               edge_color=color, connectionstyle=f"arc3,rad={rad}", ax=ax)
```

```
        nx.draw_networkx_edges(G, pos, edgelist=[(v, u)], arrowstyle="-", style="dashed",
                               edge_color=color, connectionstyle=f"arc3,rad={-rad}", ax=ax)
    else:
        nx.draw_networkx_edges(G, pos, edgelist=[(u, v)], edge_color=color, ax=ax)
    label_pos = {k: ((pos[k[0]][0] + pos[k[1]][0]) / 2, (pos[k[0]][1] + pos[k[1]][1]) / 2 + 0.1)
                 for k in edge_labels}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8, ax=ax,
label_pos=0.5)

# Set plot attributes
plt.title("Path Analysis Diagram (Cooked Grain Length as Key Trait)", fontsize=12, pad=20)
ax.set_axis_off()
plt.gca().set_facecolor("white")

# Save as PNG
plt.savefig("path_diagram.png", dpi=dpi, bbox_inches="tight", format="png")
# Save as PDF
with PdfPages("path_diagram.pdf") as pdf:
    pdf.savefig(fig, bbox_inches="tight")
plt.close()

print("Path diagrams saved to 'path_diagram.png' and 'path_diagram.pdf'")
```