

Excel Merge

Excel/CSV Merge Tool

Deze applicatie maakt het mogelijk om eenvoudig meerdere Excel- en CSV-bestanden samen te voegen op basis van gemeenschappelijke kolommen. Je kunt kolommen selecteren, duplicaten hernoemen, en de gecombineerde data opslaan.

Starten

1. **Start de applicatie** (dubbelklik op het .py-bestand of voer uit via Python).
2. Er verschijnt een venster met drie delen:
 - **Linker paneel:** Preview van het geïmporteerde bestand.
 - **Middenpaneel:** Selectie van kolommen, knoppen en instellingen.
 - **Rechter paneel:** Gecombineerde dataset.

Een bestand importeren

1. Klik op **“Import...”** om een .csv of .xlsx bestand te openen.
2. Het bestand verschijnt in het **linkerpaneel**.
3. In de lijst **“Select columns to import”** kun je de kolommen aanvinken die je wilt gebruiken.
4. Als dit het **eerste bestand** is, wordt het de basis voor verdere samenvoeging.

Bestanden samenvoegen

1. Importeer een **tweede bestand**.
2. Selecteer opnieuw de gewenste kolommen.
3. Kies bij **“Join New Key”** de kolom uit het nieuwe bestand die overeenkomt met...

4. ...de kolom in het bestaande bestand onder **“Join Existing Key”**.
5. Klik op **“Add →”** om samen te voegen.
6. Eventuele dubbele kolomnamen worden automatisch hernoemd (bv. Naam_1).

☐ Kolommen bewerken

In de kopregel van elke tabel kun je met **rechtermuisklik**:

- Een kolom **verwijderen**.
- Een kolom **hernoemen**.

☐ Resultaat opslaan

1. Klik op **“Save...”** om de gecombineerde dataset op te slaan.
2. Kies een bestandsnaam en formaat:
 - .csv voor komma-/puntkomma-gescheiden bestanden.
 - .xlsx voor Excel.

☐ Alles opnieuw beginnen

- Klik op **“Reset”** om de volledige status te wissen en opnieuw te beginnen.

i Tips

- Je kunt rijen en kolommen tellen onderaan het middenpaneel.
- Lege waarden worden weergegeven als lege velden in de preview.
- Na elke samenvoeging wordt de hoofdtabel (rechts) automatisch bijgewerkt.

Code

```
# Dependencies: pandas, openpyxl, PySide6
# Install with: pip install pandas openpyxl PySide6

import sys
import csv
import pandas as pd
```

```

from PySide6 import QtCore, QtWidgets
import pytz

# Ensure timezone usage (not strictly necessary here, but preserves your original import)
pytz.timezone("Europe/Amsterdam")

def read_csv_with_sniff(path):
    with open(path, 'rb') as f:
        raw = f.read(4096)
    try:
        text = raw.decode('utf-8', errors='replace')
    except:
        text = raw.decode('latin1', errors='replace')
    dialect = csv.Sniffer().sniff(text, delimiters=[',', ';', '\t', '|'])
    return pd.read_csv(path, delimiter=dialect.delimiter)

def read_table(path):
    if path.lower().endswith(('.xls', '.xlsx')):
        return pd.read_excel(path, engine="openpyxl")
    else:
        return read_csv_with_sniff(path)

class PandasModel(QtCore.QAbstractTableModel):
    def __init__(self, df=pd.DataFrame()):
        super().__init__()
        self._df = df

    def update(self, df):
        self.beginResetModel()
        self._df = df
        self.endResetModel()

    def rowCount(self, parent=None):
        return min(len(self._df), 100)

    def columnCount(self, parent=None):
        return len(self._df.columns)

```

```

def data(self, index, role=QtCore.Qt.DisplayRole):
    if role == QtCore.Qt.DisplayRole:
        val = self._df.iat[index.row(), index.column()]
        # Show empty string for NaN
        if pd.isna(val):
            return ""
        return str(val)
    return None

```

```

def headerData(self, section, orientation, role=QtCore.Qt.DisplayRole):
    if role == QtCore.Qt.DisplayRole:
        if orientation == QtCore.Qt.Horizontal:
            return self._df.columns[section]
        else:
            return str(section)
    return None

```

```

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Excel/CSV Merger")
        self.df = None      # main merged DataFrame
        self.new_df = None  # most recently imported DataFrame

        # — Models & TableViews —
        self.new_model = PandasModel()
        self.main_model = PandasModel()

        self.new_table = QtWidgets.QTableView()
        self.new_table.setModel(self.new_model)
        self.new_table.horizontalHeader().setContextMenuPolicy(QtCore.Qt.CustomContextMenu)
        self.new_table.horizontalHeader().customContextMenuRequested.connect(self.show_header_menu)
        self.new_table.setMinimumWidth(400)
        self.new_table.setMaximumWidth(600)

        self.main_table = QtWidgets.QTableView()
        self.main_table.setModel(self.main_model)
        self.main_table.horizontalHeader().setContextMenuPolicy(QtCore.Qt.CustomContextMenu)

```

```
self.main_table.horizontalHeader().customContextMenuRequested.connect(self.show_header_menu)
self.main_table.setMinimumWidth(400)
self.main_table.setMaximumWidth(800)
```

```
# — Buttons —
```

```
self.import_btn = QtWidgets.QPushButton("Import...")
self.add_btn    = QtWidgets.QPushButton("Add ➡")
self.export_btn = QtWidgets.QPushButton("Save...")
self.reset_btn  = QtWidgets.QPushButton("Reset")
```

```
for btn in (self.import_btn, self.add_btn, self.export_btn, self.reset_btn):
    btn.setMinimumWidth(60)
    btn.setMaximumWidth(80)
```

```
# Icons
```

```
#style = self.style()
#self.import_btn.setIcon(style.standardIcon(QtWidgets.QStyle.SP_DialogOpenButton))
#self.add_btn.setIcon  (style.standardIcon(QtWidgets.QStyle.SP_FileDialogNewFolder))
#self.export_btn.setIcon(style.standardIcon(QtWidgets.QStyle.SP_DialogSaveButton))
#self.reset_btn.setIcon (style.standardIcon(QtWidgets.QStyle.SP_BrowserReload))
```

```
# Colors
```

```
self.import_btn.setStyleSheet("background-color: #e0f7fa; color: #006064;")
self.add_btn.setStyleSheet  ("background-color: #e8f5e9; color: #1b5e20;")
self.export_btn.setStyleSheet("background-color: #fff3e0; color: #e65100;")
self.reset_btn.setStyleSheet ("background-color: #ffebee; color: #b71c1c;")
```

```
# Global disabled style
```

```
self.setStyleSheet("""
    QPushButton:disabled {
        background-color: #f0f0f0 !important;
        color: #a0a0a0  !important;
    }
""")
```

```
# — Column selector & join controls —
```

```
self.col_list      = QtWidgets.QListWidget()
self.join_new_lbl  = QtWidgets.QLabel("Join New Key:")
self.join_new_combo = QtWidgets.QComboBox()
self.join_exist_lbl = QtWidgets.QLabel("Join Existing Key:")
```

```
self.join_exist_combo = QtWidgets.QComboBox()
self.counters      = QtWidgets.QLabel("Columns: 0 | Rows: 0")

# Connect signals
self.import_btn.clicked.connect(self.select_file)
self.add_btn.clicked.connect (self.process_import)
self.export_btn.clicked.connect(self.export_result)
self.reset_btn.clicked.connect (self.reset_state)

# Initially disable until needed
self.add_btn.setEnabled(False)
self.export_btn.setEnabled(False)
self.reset_btn.setEnabled(False)
self.join_new_lbl.hide()
self.join_new_combo.hide()
self.join_exist_lbl.hide()
self.join_exist_combo.hide()

# — Middle layout (buttons, join, list, counters) —
btn_row = QtWidgets.QHBoxLayout()
btn_row.addWidget(self.import_btn)
btn_row.addWidget(self.add_btn)
btn_row.addWidget(self.export_btn)
btn_row.addWidget(self.reset_btn)

join_row = QtWidgets.QHBoxLayout()
join_row.addWidget(self.join_new_lbl)
join_row.addWidget(self.join_new_combo)
join_row.addWidget(self.join_exist_lbl)
join_row.addWidget(self.join_exist_combo)

middle_layout = QtWidgets.QVBoxLayout()
middle_layout.addLayout(btn_row)
middle_layout.addLayout(join_row)
middle_layout.addWidget(QtWidgets.QLabel("Select columns to import:"))
middle_layout.addWidget(self.col_list)
middle_layout.addStretch()
middle_layout.addWidget(self.counters)

middle_widget = QtWidgets.QWidget()
```

```

middle_widget.setLayout(middle_layout)
middle_widget.setMinimumWidth(200)

# — Splitter for three panels —
splitter = QtWidgets.QSplitter(QtCore.Qt.Horizontal)
splitter.addWidget(self.new_table)
splitter.addWidget(middle_widget)
splitter.addWidget(self.main_table)

self.setCentralWidget(splitter)
self.resize(1200, 300)

def select_file(self):
    path, _ = QtWidgets.QFileDialog.getOpenFileName(
        self, "Open File", "", "Data Files (*.csv *.xls *.xlsx)"
    )
    if not path:
        return

    try:
        self.new_df = read_table(path)
    except Exception as e:
        QtWidgets.QMessageBox.critical(
            self, "Read Error",
            f"Could not parse \"{path}\":\n{e}"
        )
        return

    # Populate columns list
    self.col_list.clear()
    for col in self.new_df.columns:
        item = QtWidgets.QListWidgetItem(col)
        item.setFlags(item.flags() | QtCore.Qt.ItemIsUserCheckable)
        item.setCheckState(QtCore.Qt.Checked)
        self.col_list.addItem(item)

    # Show or hide join-controls
    if self.df is None:
        self.join_new_lbl.hide()
        self.join_new_combo.hide()

```

```

        self.join_exist_lbl.hide()
        self.join_exist_combo.hide()
    else:
        self.join_new_lbl.show()
        self.join_new_combo.show()
        self.join_exist_lbl.show()
        self.join_exist_combo.show()
        self.join_new_combo.clear()
        self.join_new_combo.addItems(self.new_df.columns)
        self.join_exist_combo.clear()
        self.join_exist_combo.addItems(self.df.columns)

# Update preview & counters
self.new_model.update(self.new_df)
self.update_counters(self.new_df)
self.add_btn.setEnabled(True)
self.reset_btn.setEnabled(True)

def process_import(self):
    sel_cols = [
        self.col_list.item(i).text()
        for i in range(self.col_list.count())
        if self.col_list.item(i).checkState() == QtCore.Qt.Checked
    ]
    if not sel_cols:
        QtWidgets.QMessageBox.warning(self, "No columns", "Select at least one column.")
        return

    if self.df is None:
        # First file becomes main DF
        self.df = self.new_df[sel_cols].copy()
    else:
        # Merge into existing
        key_new = self.join_new_combo.currentText()
        key_exist = self.join_exist_combo.currentText()
        # Strip spaces from keys for proper matching
        key_new_stripped = key_new.strip()
        key_exist_stripped = key_exist.strip()
        cols_to_take = [key_new] + [c for c in sel_cols if c != key_new]
        df2 = self.new_df[cols_to_take].copy()

```



```

# Rename duplicates
rename_map = {}
for col in sel_cols:
    if col == key_new: continue
    if col in self.df.columns:
        i = 1
        while f"{col}_{i}" in self.df.columns:
            i += 1
        rename_map[col] = f"{col}_{i}"
df2.rename(columns=rename_map, inplace=True)

merged = pd.merge(
    self.df, df2,
    how='outer',
    left_on=key_exist_stripped,
    right_on=key_new_stripped,
    suffixes=(None, None)
)
if key_new_stripped != key_exist_stripped:
    merged[key_exist_stripped] = merged[key_exist_stripped].fillna(merged[key_new_stripped])
    merged.drop(columns=[key_new_stripped], inplace=True)
self.df = merged

```

```

# Update main preview & counters

```

```

self.main_model.update(self.df)
self.update_counters(self.df)
self.export_btn.setEnabled(True)

```

```

# Clear new_df state

```

```

self.new_df = None
self.new_model.update(pd.DataFrame())
self.add_btn.setEnabled(False)
self.reset_btn.setEnabled(True)

```

```

def export_result(self):

```

```

    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Save Result", "results.csv", "CSV Files (*.csv);;Excel Files (*.xlsx)"
    )

```

```

    if not path:

```

```
        return
    if path.lower().endswith(('.xls', '.xlsx')):
        self.df.to_excel(path, index=False)
    else:
        self.df.to_csv(path, index=False)
    QtWidgets.QMessageBox.information(self, "Exported", f"Saved to {path}")
```

```
def reset_state(self):
    self.df = None
    self.new_df = None
    self.main_model.update(pd.DataFrame())
    self.new_model.update(pd.DataFrame())
    self.col_list.clear()
    self.counters.setText("Columns: 0 | Rows: 0")
    self.add_btn.setEnabled(False)
    self.export_btn.setEnabled(False)
    self.reset_btn.setEnabled(False)
    self.join_new_lbl.hide()
    self.join_new_combo.hide()
    self.join_exist_lbl.hide()
    self.join_exist_combo.hide()
```

```
def update_counters(self, df=None):
    if df is None:
        df = pd.DataFrame()
    cols, rows = (df.shape if not df.empty else (0, 0))
    self.counters.setText(f"Columns: {cols} | Rows: {rows}")
```

```
def get_current_df(self):
    # Called by the header-menu actions
    current = self.sender().parent() # the QHeaderView
    df = None; which = None
    if self.new_table.horizontalHeader() is current:
        df, which = self.new_df, 'new_df'
    else:
        df, which = self.df, 'df'
    return df, which
```

```
def show_header_menu(self, pos):
    header = self.sender()
```

```

logical_index = header.logicalIndexAt(pos)
if logical_index < 0:
    return
menu = QtWidgets.QMenu(self)
delete_action = menu.addAction("Delete")
rename_action = menu.addAction("Rename")
action = menu.exec_(header.mapToGlobal(pos))
if action == delete_action:
    self.delete_column(logical_index)
elif action == rename_action:
    self.rename_column(logical_index)

def delete_column(self, col_idx):
    df, which = self.get_current_df()
    if df is None or df.empty:
        return
    if len(df.columns) == 1:
        QtWidgets.QMessageBox.warning(self, "Cannot Delete", "At least one column must remain.")
        return
    col_name = df.columns[col_idx]
    df.drop(columns=[col_name], inplace=True)
    if which == 'new_df':
        self.new_model.update(df)
    else:
        self.main_model.update(df)
    self.update_counters(df)

def rename_column(self, col_idx):
    df, which = self.get_current_df()
    if df is None or df.empty:
        return
    old = df.columns[col_idx]
    new_name, ok = QtWidgets.QInputDialog.getText(self, "Rename Column", f"Rename '{old}' to:", text=old)
    if not ok or not new_name.strip():
        return
    new = new_name.strip()
    if new == old or new in df.columns:
        QtWidgets.QMessageBox.warning(self, "Invalid Name", "Choose a unique, non-empty name.")
        return
    df.rename(columns={old: new}, inplace=True)

```

```
if which == 'new_df':  
    self.new_model.update(df)  
else:  
    self.main_model.update(df)  
self.update_counters(df)
```

```
if __name__ == '__main__':  
    app = QtWidgets.QApplication(sys.argv)  
    window = MainWindow()  
    window.show()  
    sys.exit(app.exec())
```

Revision #2

Created 30 June 2025 19:01:51 by Max

Updated 30 June 2025 19:06:09 by Max