```python
from statsmodels.regression.linear_model import RegressionResultsWrapper
from statsmodels.tsa.vector_ar.vecm import VECMResults
from linearmodels.panel.results import PanelEffectsResults
import statsmodels.api as sm

from src.utils import get_stars


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def _get_statsmodels_ols_summary(mod):
    df = pd.DataFrame(pd.concat([mod.params, mod.pvalues, mod.tvalues], axis=1))
    df.columns = ['coef', 'pval', 'stat']

    df_conf = mod.conf_int()
    df_conf.columns = ['conf_lower', 'conf_upper']

    endog_name = mod.model.endog_names
    df = df.join(df_conf)

    df_info = pd.DataFrame([], columns=df.columns)
    df_info.loc['R^2'] = list([mod.rsquared] * df.shape[1])
    df_info.loc['R^2 adj.'] = list([mod.rsquared_adj] * df.shape[1])
    df_info.loc['N'] = list([mod.nobs] * df.shape[1])

    df = pd.concat([df, df_info],)
    df['is_info'] = list([False] * (len(df) - len(df_info))) + list([True] * len(df_info))
    df['is_print'] = ~df.is_info

    return df, endog_name




def _get_linearmodels_pols_summary(mod):
    df = pd.DataFrame(pd.concat([mod.params, mod.pvalues, mod.tstats], axis=1))
    df.columns = ['coef', 'pval', 'stat']

    df_conf = mod.conf_int()
    df_conf.columns = ['conf_lower', 'conf_upper']

    endog_name = str(mod.model.dependent.dataframe.columns[0])
    df = df.join(df_conf)

    df_info = pd.DataFrame([], columns=df.columns)
    df_info.loc['R^2 between'] = list([mod.rsquared_between] * df.shape[1])
    df_info.loc['R^2 within'] = list([mod.rsquared_within] * df.shape[1])
    df_info.loc['Entity effects'] = list([mod.model.entity_effects] * df.shape[1])
    df_info.loc['N'] = list([mod.nobs] * df.shape[1])
    df_info.loc['N entity'] = list([len(set([i[0] for i in mod.fitted_values.index]))] * df.shape[1])
    df_info.loc['N time'] = list([len(set([i[1] for i in mod.fitted_values.index]))] * df.shape[1])

    df = pd.concat([df, df_info],)
    df['is_info'] = list([False] * (len(df) - len(df_info))) + list([True] * len(df_info))
    df['is_print'] = ~df.is_info

    return df, endog_name

def _get_statmodels_vecm_summary(mod, endog_index: 0, sig: float = .05):
    endog_name = mod.model.endog_names[endog_index]

    df = pd.DataFrame(mod.summary().tables[0].data).iloc[1:].set_index(0)
    df.index.name = ''
    df.columns = ['coef', 'stderr', 'stat', 'pval', 'conf_lower', 'conf_upper']
    df = df.astype(float)
    df = df[['coef', 'pval', 'stat', 'conf_lower', 'conf_upper']]

    df_info = pd.DataFrame([], columns=df.columns)
    df_info.loc['Coint. rank'] = list([mod.model.coint_rank] * df.shape[1])
    df_info.loc['N lags'] = list([mod.k_ar] * df.shape[1])
    df_info.loc['N'] = list([mod.nobs] * df.shape[1])


    h0: resid autocorrelation is zero up to lag 10
    p, s = mod.test_whiteness().pvalue, mod.test_whiteness().crit_value
    df_info.loc['Whiteness'] = list([str(bool(~(p<=sig))), p, s, 0, 0, ])


    df = pd.concat([df, df_info])
    df['is_info'] = list([False] * (len(df) - len(df_info))) + list([True] * len(df_info))
    df['is_print'] = list([True] * (len(df) - len(df_info))) + list([False] * (len(df_info)-2)) + list([True] * 2)

    return df , endog_name


def get_statsmodels_summary(lst_mods, cols_out: str = 'print', vecm_endog_index: int = 0, seperator: str = "\n",
                            thresh_sig: float = .05, is_filt_sig: bool = False, n_round: int = 3):
    """
    Prints summary table for statmodels regression models
    :param lst_mods:
    :param cols_out:
    :param vecm_endog_index:
    :param seperator:
    :param thresh_sig:
    :param is_filt_sig:
    :param n_round:
    :return:
    """
    lst_dfs, lst_endog_names = [], []
    for idx, mod in enumerate(lst_mods):

        if type(mod) == RegressionResultsWrapper:
            df, endog_name = _get_statsmodels_ols_summary(mod)

        elif type(mod) == VECMResults:
            df, endog_name = _get_statmodels_vecm_summary(mod, vecm_endog_index, sig=thresh_sig)

        elif type(mod) == PanelEffectsResults:
            df, endog_name = _get_linearmodels_pols_summary(mod)

        else:
            raise KeyError(f"{type(mod)} not specified")

        if endog_name in lst_endog_names:
                endog_name += f"_{idx}"
        lst_endog_names.append(endog_name)


        # print output
        df['star'] = df['pval'].apply(lambda x: get_stars(x))
        df['print'] = df['coef']
        df['print'] = df.coef.round(n_round).astype(str) + " " + df.star.astype(str) + seperator + "[" + df.stat.round(n_round).astype(str) + "]"
        df.loc[~df['is_print'], 'print'] = df.loc[~df.is_print, 'coef'].round(n_round).astype(str)

        # significance thresh
        df['is_significant'] = (df['pval'] <= thresh_sig)

        cols = [list(df.columns), list([endog_name] * df.shape[1])]
        df.columns = pd.MultiIndex.from_tuples(list(map(tuple, zip(*cols))))

        lst_dfs.append(df)

    out = pd.concat([df for df in lst_dfs], axis=1, join='outer').sort_index(axis=1)
    is_sig_filt = (out['is_significant'].sum(axis=1) > 0).values
    out['is_info_sum'] = out['is_info'].sum(axis=1) > 0

    if is_filt_sig:
        out = out.loc[(is_sig_filt + out['is_info_sum'] > 0)]
    out = out.sort_index().sort_values('is_info_sum')

    out = out[cols_out]

    return out


def get_dfbetas(X: np.array, resid: np.array):
    """
    Computes dfbetas
    :param X:
    :param resid:
    :return:
    """
    assert X.shape[0] == resid.shape[0], "X and resid do not correspond"
    lst_dfbetas = []

    H = X @ np.linalg.inv(X.T @ X) @ X.T
    H_diag = np.diagonal(H)
    for idx in range(X.shape[0]):
        x_i = X[idx]
        e_i = resid[idx]

        lst_dfbetas.append(
            (np.linalg.inv(X.T @ X) @ x_i[None].T @ e_i[None]) / (1 - H_diag[idx])
        )
    return np.array(lst_dfbetas)


def get_cooks_distance(X: np.array, resid: np.array, flt_largest_perc: float = 97.5):
    """
    Calculates Cook's distance
    :param X:
    :param resid:
    :param flt_largest_perc:
    :return:
    """
    n, p = len(resid), np.linalg.matrix_rank(X)

    s_2 = resid.T @ resid / (n - p)
    H = X @ np.linalg.inv(X.T @ X) @ X.T
    H_diag = np.diagonal(H)

    lst_cooks_dist = []
    for idx in range(X.shape[0]):
        d_i = resid[idx] ** 2 / p * s_2 * (H_diag[idx] / (1 - H_diag[idx]) ** 2)
        lst_cooks_dist.append(d_i)
    arr_cook_dist = np.array(lst_cooks_dist)
    filt_percent = arr_cook_dist >= np.percentile(arr_cook_dist, flt_largest_perc)
    return arr_cook_dist, filt_percent


def get_fig_subplots(n_plots: int = 1, n_cols: int = 1, figsize: tuple = None, **kwargs):
    if figsize is None:
        figsize = tuple(plt.rcParams["figure.figsize"])
    n_rows = int(np.ceil(n_plots/n_cols))
    fig, ax = plt.subplots(n_rows, n_cols, figsize=(figsize[0] * n_rows, figsize[1] * n_cols), **kwargs)
    if n_plots == 1 and n_cols == 1:
        return fig, ax
    else:
        ax = ax.ravel()[:n_plots]
    return fig, ax


def get_multiple_vecm_irfs(lst_vecms, idx_vecm: tuple = (0,1), irf_periods: int = 5, dict_titles: dict = None, **kwargs):
    fig, axes = get_fig_subplots(len(lst_vecms), **kwargs)
    for idx, ax in enumerate(axes):
        irf = lst_vecms[idx].irf(periods=irf_periods)

        ax.plot(irf.irfs[:, *idx_vecm], color='blue', label='irf')
        ax.fill_between(range(len(irf.irfs)),
                        irf.irfs[:, idx_vecm[0], idx_vecm[1]] + 1.96 * irf.stderr()[:, idx_vecm[0], idx_vecm[1]],
                        irf.irfs[:, idx_vecm[0], idx_vecm[1]] - 1.96 * irf.stderr()[:, idx_vecm[0], idx_vecm[1]],
                        alpha=.3, color='grey', linestyle='dashed', label='90% conf.')

        ax.plot(list([0] * irf.irfs.shape[0]), color='black')

        n1, n2 = lst_vecms[idx].names[idx_vecm[1]], lst_vecms[idx].names[idx_vecm[0]]
        # print(v.names)
        if dict_titles is not None:
            try:
                n1 = dict_titles[n1]
                n2 = dict_titles[n2]
            except Exception as e:
                n1, n2 = lst_vecms[idx].names[idx_vecm[1]], lst_vecms[idx].names[idx_vecm[0]]

        ax.set_title(f"{n1} -> {n2}",)
        ax.legend()

    fig.tight_layout()
    return fig
```