Data Science I: Homework 1 Python Code

Neal Kuperman

January 15, 2026

Contents

ch. 1.

Helper Functions

1.1 Imports

```
import ISLP
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import statsmodels.api as smf
from IPython.display import display
from lin_reg_plots import LinearRegDiagnostic
import pickle
from ucimlrepo import fetch_ucirepo
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SequentialFeatureSelector
```

1.2 Summarize Function

```
    def summarize(model, vars = [], verbose = True):
        if isinstance(model.params, np.ndarray):
            params = model.params
            if len(vars) != (len(params) - 1):
                if verbose:
                    print("Warning: The number of variables does not match
                        the number of parameters")
                vars = ["intercept"] + [f"x{i}" for i in range(len(params)
                    - 1)]
        else:
            params = model.params.values
            vars = model.params.index.to_list()

        tvalues = np.round(model.tvalues, 4)
        pvalues = np.round(model.pvalues, 4)
        std_err = np.round(model.bse, 4)

        param_summary = pd.DataFrame(index=vars)
        param_summary["coef"] = params
        param_summary["t value"] = tvalues
        param_summary["p value"] = pvalues
        param_summary["std err "] = std_err


        r_squared = np.round(model.rsquared, 4)
        F_stat = np.round(model.fvalue, 4)
        # model_summary = pd.DataFrame(columns=["R-squared", "F-statistic
            "])
        model_summary = pd.DataFrame({"R-squared": [r_squared], "F-
            statistic": [F_stat]}, index = ["value"])

        return param_summary, model_summary
```

1.3 Leverage Functions

```
    def calc_leverage(x):
        """
        Observations with high leverage have an unusual value for xi.
        High leverage observations tend to have a sizable impact on the
        estimated regression line.

        SLR:  hi = 1/n + (xi - x_bar)^2 / sum((xi - x_bar)^2)
        MLR:  Hii = diag(H), where H = X^T (X^T X)^-1 X
        """
        H = x @ np.linalg.inv(x.T @ x) @ x.T
        leverage_points = np.diag(H)
        return leverage_points

    def find_high_leverage_points(x):
        """
        Observations with high leverage have an unusual value for xi.
        hii > 2p/n
            p = number of predictors
            n = number of observations
        """
        leverage_points = calc_leverage(x)
        high_leverage_points = leverage_points > 2 * x.shape[1]/x.shape[0]
        index = np.where(high_leverage_points)[0]
        return index, leverage_points[index]
```

1.4 Cook's Distance

```
    def calc_cooks_distance(x, y):
        """
        Cook's distance is a measure of the influence of an observation
        on the estimated regression coefficients.

        D_i = (e_i^2 / (p * MSE)) * (h_ii / (1 - h_ii)^2)

        where:
            e_i = residual (y_i - y_hat_i)
            h_ii = leverage (diagonal of hat matrix)
            p = number of predictors (including intercept)
            MSE = mean squared error = SS_res / (n - p)
        """
        n, p = x.shape

        # Fit model
        beta = np.linalg.inv(x.T @ x) @ x.T @ y
        y_hat = x @ beta
        residuals = y - y_hat

        # MSE
        SS_res = np.sum(residuals**2)
        MSE = SS_res / (n - p)

        # Leverage
        h = calc_leverage(x)

        # Cook's distance
        cooks_d = (residuals**2 / (p * MSE)) * (h / (1 - h)**2)

        return cooks_d
```

ch. 2.

ISLP 3.10

2.1 Load Data

```
carseats = ISLP.load_data('Carseats')

quant_cols = carseats.select_dtypes(include=['number']).columns
cat_cols = carseats.select_dtypes(include=['category']).columns

# convert categorical columns to dummy variables. Creates a new column for each category level for each categorical column.
carseats = pd.get_dummies(carseats, columns=cat_cols, dtype=float)
```

2.2 (a) Multiple Regression: Sales $\sim$ Price + Urban + US

```
    # Must add an intercept column when using statsmodels.api.OLS
    X = carseats[["Price", "Urban_Yes", "US_Yes"]]
    X.insert(0, 'Intercept', 1.0)
    y = carseats["Sales"]

    model = smf.OLS(y, X).fit()
    print(model.summary())
```

2.3 (e) OLS with Price and US_Yes Only

```
    X = carseats[["Price", "US_Yes"]]
    X.insert(0, 'Intercept', 1.0)
    y = carseats["Sales"]

    model = smf.OLS(y, X).fit()
    print(model.summary())
    # print(model.summary().as_latex())
```

2.4 (g) Confidence Intervals

```
model.conf_int(alpha=0.05)
```

2.5 (h) High Leverage Points and Cook's Distance

```
    index, high_leverage_points = find_high_leverage_points(X.values)
    lvg_df = pd.DataFrame({"index": index, "high_leverage_points":
        high_leverage_points})
    # print(lvg_df.to_latex(index=False))

    # Get influence measures
    influence = model.get_influence()
    cooks_d = influence.cooks_distance[0]  # [0] is the values, [1] is p-
        values


    # Plot Cook's D
    plt.stem(range(len(cooks_d)), cooks_d, markerfmt=",")
    plt.xlabel('Observation')
    plt.ylabel("Cook's Distance")
    plt.savefig("../images/3_10_h_cooks_d.png")
    plt.show()
```

ch. 3.

ISLP 3.14

3.1 (a) Generate Data

```
    np.random.seed(5)
    rng = np.random.default_rng(10)
    x1 = rng.uniform(0, 1, size=100)
    x2 = 0.5 * x1 + rng.normal(size=100) / 10
    y = 2 + 2 * x1 + 0.3 * x2 + rng.normal(size=100)
```

3.2 (b) Correlation Between x1 and x2

```
    # Manual calculation of correlation
    x1_mean = np.mean(x1)
    x2_mean = np.mean(x2)

    numerator = np.sum((x1 - x1_mean) * (x2 - x2_mean))
    denominator = np.sqrt(np.sum((x1 - x1_mean)**2) * np.sum((x2 - x2_mean)
        **2))

    correlation = numerator / denominator
    print(f"Correlation between x1 and x2: {correlation:.3f}")

    plt.scatter(x1, x2)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('Scatterplot of x1 and x2')
    plt.show()
```

3.3 (c) OLS with x1 and x2

```
    intercept = np.ones(len(x1))
    X = np.column_stack((intercept, x1, x2))
    model = smf.OLS(y, X).fit()
    print(model.summary())
    # print(model.summary().as_latex())
```

3.4 (d) OLS with x1 Only

```
    intercept = np.ones(len(x1))
    X = np.column_stack((intercept, x1))
    model = smf.OLS(y, X).fit()
    print(model.summary())
    # print(model.summary().as_latex())
```

3.5 (e) OLS with x2 Only

```
    intercept = np.ones(len(x1))
    X = np.column_stack((intercept, x2))
    model = smf.OLS(y, X).fit()
    print(model.summary())
    # print(model.summary().as_latex())
```

3.6 (g) New Observation Analysis

```
    x1_new = np.concatenate([x1, [0.1]])
    x2_new = np.concatenate([x2, [0.8]])
    y_new = np.concatenate([y, [6]])

    intercept = np.ones(len(x1_new))
    X_full = np.column_stack((intercept, x1_new, x2_new))
    X_x1 = np.column_stack((intercept, x1_new))
    X_x2 = np.column_stack((intercept, x2_new))

    model_full = smf.OLS(y_new, X_full).fit()
    model_x1 = smf.OLS(y_new, X_x1).fit()
    model_x2 = smf.OLS(y_new, X_x2).fit()

    print("Full Model")
    print("="*60)
    param_summary_full, model_summary_full = summarize(model_full, verbose=
        False)
    display(param_summary_full)
    display(model_summary_full)

    print("\nx1 Model")
    print("="*60)
    param_summary_x1, model_summary_x1 = summarize(model_x1, verbose=False)
    display(param_summary_x1)
    display(model_summary_x1)

    print("\nx2 Model")
    print("="*60)
    param_summary_x2, model_summary_x2 = summarize(model_x2, verbose=False)
    display(param_summary_x2)
    display(model_summary_x2)

    # Check for high leverage and influential points
    index, high_leverage_points = find_high_leverage_points(X_full)
    print(index)
    print(high_leverage_points)

    influential_stats = model_full.get_influence()
    cooks_D = influential_stats.cooks_distance[0]
    influential_pts_index = np.where(cooks_D > 1)[0]
    influential_pts_vals = cooks_D[influential_pts_index]

    print(influential_pts_index)
    print(influential_pts_vals)

    cls = LinearRegDiagnostic(model_full)
    vif, fig, ax = cls()
    print(vif)
```

ch. 4.

ISLP 3.15

4.1 (a) Simple Linear Regression for Each Predictor

```
    bos = ISLP.load_data('Boston')

    # Get predictor columns (excluding 'crim')
    predictors = [col for col in bos.columns if col != 'crim']
    n_predictors = len(predictors)

    # Create subplot grid
    n_cols = 3
    n_rows = (n_predictors + n_cols - 1) // n_cols
    fig_1, axes_1 = plt.subplots(n_rows, n_cols, figsize=(16, 3*n_rows))
    axes_1 = axes_1.flatten()

    fig_2, axes_2 = plt.subplots(n_rows, n_cols, figsize=(16, 3*n_rows))
    axes_2 = axes_2.flatten()

    models = {}
    r_squared = []

    for i, col_name in enumerate(predictors):
        if col_name == 'crim':
            continue
        X = bos[col_name].values
        intercept = np.ones(len(X))
        X = np.column_stack((intercept, X))
        model = smf.OLS(bos['crim'], X).fit()
        models[col_name] = model
        predictions = model.predict(X)
        residual = bos['crim'] - predictions
        r_squared.append(model.rsquared)

        axes_1[i].scatter(bos[col_name], bos['crim'], alpha=0.5)
        axes_1[i].plot(bos[col_name].sort_values(), predictions[bos[
            col_name].argsort()], color='red')
        axes_1[i].set_xlabel(col_name, fontsize=14)
        axes_1[i].set_ylabel('crim', fontsize=14)
        axes_1[i].set_title(f'crim vs {col_name}', fontsize=14)

        axes_2[i].scatter(bos[col_name], residual, alpha=0.5)
        axes_2[i].set_xlabel(col_name, fontsize=14)
```

```
        axes_2[i].set_ylabel('residual', fontsize=14)
        axes_2[i].set_title(f'residual - {col_name}', fontsize=14)

    fig_1.tight_layout(pad=1.5, h_pad=2, w_pad=1)
    fig_1.savefig('../images/3_15_a_scatter.png', dpi=300, bbox_inches='
        tight')

    fig_2.tight_layout(pad=1.5, h_pad=2, w_pad=1)
    fig_2.savefig('../images/3_15_a_residuals.png', dpi=300, bbox_inches='
        tight')

    plt.show()

    r_squared_df = pd.DataFrame({'predictor': predictors, 'r_squared':
        r_squared})
    r_squared_df = r_squared_df.sort_values(by='r_squared', ascending=False
        )
    print(r_squared_df)
```

4.2 (b) Multiple Regression with All Predictors

```
    X = bos[predictors]
    X.insert(0, 'Intercept', 1.0)
    y = bos['crim']
    model = smf.OLS(y, X).fit()
    print(model.summary())

    param_summary, model_summary = summarize(model, verbose=False)
    display(param_summary)
    display(model_summary)
```

4.3 (d) Non-linear Associations (Polynomial Regression)

```
    n_cols = 3
    n_rows = (n_predictors + n_cols - 1) // n_cols
    fig_1, axes_1 = plt.subplots(n_rows, n_cols, figsize=(16, 3*n_rows))
    axes_1 = axes_1.flatten()

    fig_2, axes_2 = plt.subplots(n_rows, n_cols, figsize=(16, 3*n_rows))
    axes_2 = axes_2.flatten()

    models = {}
    r_squared = []

    for i, col_name in enumerate(predictors):
        if col_name == 'crim':
            continue
        X = bos[col_name].values
        X2 = X**2
        X3 = X**3
        intercept = np.ones(len(X))
        X = np.column_stack((intercept, X, X2, X3))
        _df = pd.DataFrame(X)
        _df.columns = ['intercept', col_name, f'{col_name}^2', f'{col_name
            }^3']
        model = smf.OLS(bos['crim'], _df).fit()
        models[col_name] = model
        predictions = model.predict(X)
        residual = bos['crim'] - predictions
        r_squared.append(model.rsquared)

        print(f"Model for {col_name}:")
        print("="*60)
        param_summary, model_summary = summarize(model, verbose=False)
        display(param_summary)
        display(model_summary)
        print("\n")

        axes_1[i].scatter(bos[col_name], bos['crim'], alpha=0.5)
        axes_1[i].plot(bos[col_name].sort_values(), predictions[bos[
            col_name].argsort()], color='red')
        axes_1[i].set_xlabel(col_name, fontsize=14)
        axes_1[i].set_ylabel('crim', fontsize=14)
        axes_1[i].set_title(f'crim vs {col_name}', fontsize=14)

        axes_2[i].scatter(bos[col_name], residual, alpha=0.5)
        axes_2[i].set_xlabel(col_name, fontsize=14)
        axes_2[i].set_ylabel('residual', fontsize=14)
        axes_2[i].set_title(f'residual - {col_name}', fontsize=14)

    fig_1.tight_layout(pad=1.5, h_pad=2, w_pad=1)
```

```
    fig_1.savefig('../images/3_15_d_scatter.png', dpi=300, bbox_inches='
        tight')

    fig_2.tight_layout(pad=1.5, h_pad=2, w_pad=1)
    fig_2.savefig('../images/3_15_d_residuals.png', dpi=300, bbox_inches='
        tight')

    plt.tight_layout()
    plt.show()

    r_squared_df = pd.DataFrame({'predictor': predictors, 'r_squared':
        r_squared})
    r_squared_df = r_squared_df.sort_values(by='r_squared', ascending=False
        )
    print(r_squared_df)
    # r_squared_df.to_latex(index = False)
```

ch. 5.

ESL 3.17

5.1 Load and Prepare Data

```
    spambase = fetch_ucirepo(id=94)
    scaler = StandardScaler()

    X = spambase.data.features
    y = spambase.data.targets

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42
    )

    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

5.2 OLS, Ridge, and Lasso Regression

```
    models = {
        'LS': {
            'model': LinearRegression(),
            'param_grid': {}
        },
        'ridge': {
            'model': Ridge(),
            'param_grid': {
                'alpha': np.concatenate([np.arange(0.005,10, 0.05), np.
                    arange(10,2000,10)]),
                "max_iter": [1000]
            }
        },
        'lasso': {
            'model': Lasso(),
            'param_grid': {
                'alpha': np.concatenate([np.arange(0.01,.1, 0.001), np.
                    arange(1,2000,10)]),
                'max_iter': [1000]
            }
        },
    }

    results = {
        "name": [], "model": [], "intercept": [],
        "coefficients": [], "test_error": [], "test_R2": []
    }

    for name, model in models.items():
        print()
        print(name)
        print("="*60)
        opt_param = {}

        if model["param_grid"]:
            grid_search = GridSearchCV(
                model["model"],
                param_grid=model["param_grid"],
                cv=5,
                scoring='neg_mean_squared_error'
            )
            grid_search.fit(X_train_scaled, y_train)
            opt_param = {
                "alpha": grid_search.best_params_['alpha'],
                "max_iter": grid_search.best_params_['max_iter']
            }

        model = model["model"].set_params(**opt_param)
```

```
        model.fit(X_train_scaled, y_train)

        test_pred = model.predict(X_test_scaled)
        test_error = mean_squared_error(y_test, test_pred)
        test_R2 = r2_score(y_test, test_pred)

        print(test_R2)

        results["name"].append(name)
        results["model"].append(model)
        results["intercept"].append(model.intercept_[0])
        results["coefficients"].append(model.coef_.flatten())
        results["test_error"].append(test_error)
        results["test_R2"].append(test_R2)

    # Create coefficients DataFrame
    coefs = np.column_stack([arr for arr in results["coefficients"]])
    coefs = np.vstack([results["intercept"], coefs])

    cols = X.columns.to_list()
    cols = ['intercept'] + cols
    model_names = results["name"]

    df = pd.DataFrame(coefs, index=cols, columns=model_names)
```

5.3 Best Subset Selection

```
    def best_subset_selection(X, y, X_train, X_test, y_train, y_test, rerun
        =False):
        best_subset_models = {}
        results = {}
        linear_models = {
            "model": [],
            "coefs": [],
            "intercept": [],
            "score": []
        }

        if rerun:
            for i in range(1, len(X.columns)):
                print(i)
                sfs = SequentialFeatureSelector(
                    LinearRegression(),
                    n_features_to_select=i,
                    direction='forward',
                )
                sfs.fit(X_train, y_train)

                # Select features from TRAIN data, fit on TRAIN
                selected_train = X_train[:, sfs.get_support()]
                selected_test = X_test[:, sfs.get_support()]

                selected_cols = X.columns[sfs.get_support()]
                df_train = pd.DataFrame(selected_train, columns=
                    selected_cols)
                df_test = pd.DataFrame(selected_test, columns=selected_cols
                    )

                lin_mod = LinearRegression()
                lin_mod.fit(df_train, y_train)

                best_subset_models[i] = sfs
                results[i] = sfs.get_support()
                linear_models["model"].append(lin_mod)
                linear_models["coefs"].append(lin_mod.coef_)
                linear_models["intercept"].append(lin_mod.intercept_)
                linear_models["score"].append(lin_mod.score(df_test, y_test
                    ))

            results = pd.DataFrame(results)
            results.index.name = 'Feature'

            save_dict = {
                "linear_models": linear_models,
                "results": results,
```

```
                "best_subset_models": best_subset_models
            }

            with open("HW_1_ESL_3_17.pkl", "wb") as f:
                pickle.dump(save_dict, f)

        else:
            with open('HW_1_ESL_3_17.pkl', 'rb') as f:
                save_dict = pickle.load(f)
            results = save_dict["results"]
            linear_models = save_dict["linear_models"]
            best_subset_models = save_dict["best_subset_models"]

        plt.scatter(range(1, len(linear_models["score"])+1), linear_models[
            "score"])
        plt.savefig("../images/ESL_3_17_best_subset_score.png")
        plt.show()
        return results, linear_models, best_subset_models

    # Run best subset selection
    best_subset_results, best_subset_linear_models, best_subset_models = \
        best_subset_selection(X, y, X_train, X_test, y_train, y_test, rerun
            =False)

    best_subset_idx = best_subset_linear_models["score"].index(
        max(best_subset_linear_models["score"])
    )
    best_subset_model = best_subset_models[best_subset_idx]

    col_mask = best_subset_model.get_support()
    cols = X.columns[col_mask]

    # Select features from TRAIN data, fit on TRAIN
    selected_train = X_train_scaled[:, col_mask]
    selected_test = X_test_scaled[:, col_mask]

    # Add intercept
    selected_train_with_intercept = smf.add_constant(selected_train)
    selected_test_with_intercept = smf.add_constant(selected_test)

    # Fit OLS model
    model = smf.OLS(y_train, selected_train_with_intercept).fit()

    # Create best subset df
    all_indices = df.index.to_list()
    best_subset_df = pd.DataFrame({"best_subset": model.params[1:].values},
         index=cols)

    best_subset_df_all_indices = pd.DataFrame(
        {"best_subset": [0.0]*len(all_indices)},
```

```
        index=all_indices
    )

    shared_idx = best_subset_df.index.intersection(
        best_subset_df_all_indices.index)
    best_subset_df_all_indices.loc[shared_idx, "best_subset"] = \
        best_subset_df.loc[shared_idx, "best_subset"]
    best_subset_df_all_indices.loc["intercept", "best_subset"] = model.
        params.iloc[0]

    # Add best subset model to df of coefficients
    df["best_subset"] = best_subset_df_all_indices["best_subset"]

    # Add best subset model info to results dictionary
    test_pred = model.predict(selected_test_with_intercept)
    test_error = mean_squared_error(y_test, test_pred)
    test_R2 = r2_score(y_test, test_pred)

    results["name"].append("best subset")
    results["model"].append(model)
    results["intercept"].append(model.params.iloc[0])
    results["coefficients"].append(model.params[1:].values)
    results["test_error"].append(test_error)
    results["test_R2"].append(test_R2)
```

5.4 Coefficient Comparison Plot

```
    # Split features into 10 groups for readability
    n_features = len(df)
    chunk_size = (n_features + 9) // 10

    fig, axes = plt.subplots(5, 2, figsize=(14, 20))
    axes = axes.flatten()

    for i, ax in enumerate(axes):
        start = i * chunk_size
        end = min((i + 1) * chunk_size, n_features)

        df.iloc[start:end].plot(kind='bar', ax=ax, width=0.8, legend=False)

        ax.set_ylabel('Coefficient Value')
        ax.axhline(y=0, color='black', linestyle='-', linewidth=0.5)
        ax.tick_params(axis='x', rotation=45)

        for label in ax.get_xticklabels():
            label.set_ha('right')

    axes[0].set_title('Coefficients by Feature')

    handles, labels = axes[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc='lower center', ncol=len(labels),
        bbox_to_anchor=(0.5, -0.02))

    plt.tight_layout()
    fig.subplots_adjust(bottom=0.08)

    plt.savefig("../images/ESL_3_17_coefficients.png", bbox_inches='tight')
    plt.show()

    # Print results
    print(df)
    print(results["test_R2"])
    print(results["test_error"])
```