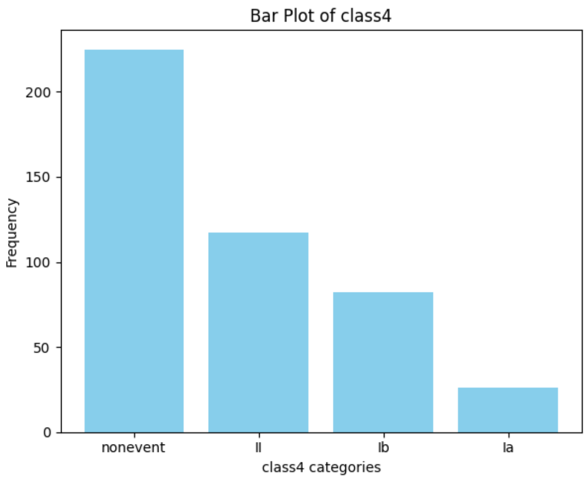
Problem1

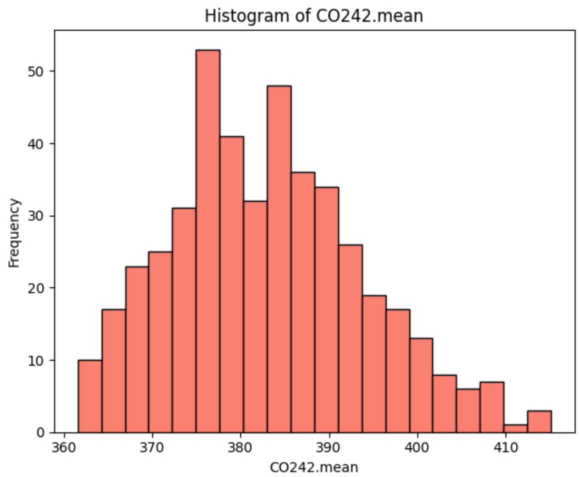
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

path = './data/'
# problem1
# loda data
df = pd.read_csv(path+"train.csv")

# data info
print(df.head())
print(df.info())
print(df.describe())

# drop the columns "id" and "partlybad"
df = df.drop(columns=["id", "partlybad"])

print(df.columns)

# caculate mean and std of t84.mean
selected_columns = df[["T84.mean", "UV_A.mean", "CS.mean"]]
print(selected_columns.describe())

t84_array = df["T84.mean"].values
mean_t84 = np.mean(t84_array)
std_t84 = np.std(t84_array)
print("the mean of t84.mean",mean_t84)
print("the standard deviation of t84.mean",std_t84)

# task d
class4_counts = df["class4"].value_counts()
co242_data = df["CO242.mean"]

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
# Bar plot of class4
axes[0].bar(class4_counts.index, class4_counts.values, color='skyblue')
axes[0].set_title("Bar Plot of class4")
axes[0].set_xlabel("class4 categories")
axes[0].set_ylabel("Frequency")

# Histogram of CO242.mean
axes[1].hist(co242_data, bins=20, color='salmon', edgecolor='black')
axes[1].set_title("Histogram of CO242.mean")
axes[1].set_xlabel("CO242.mean")
axes[1].set_ylabel("Frequency")

plt.tight_layout()
plt.show()

```

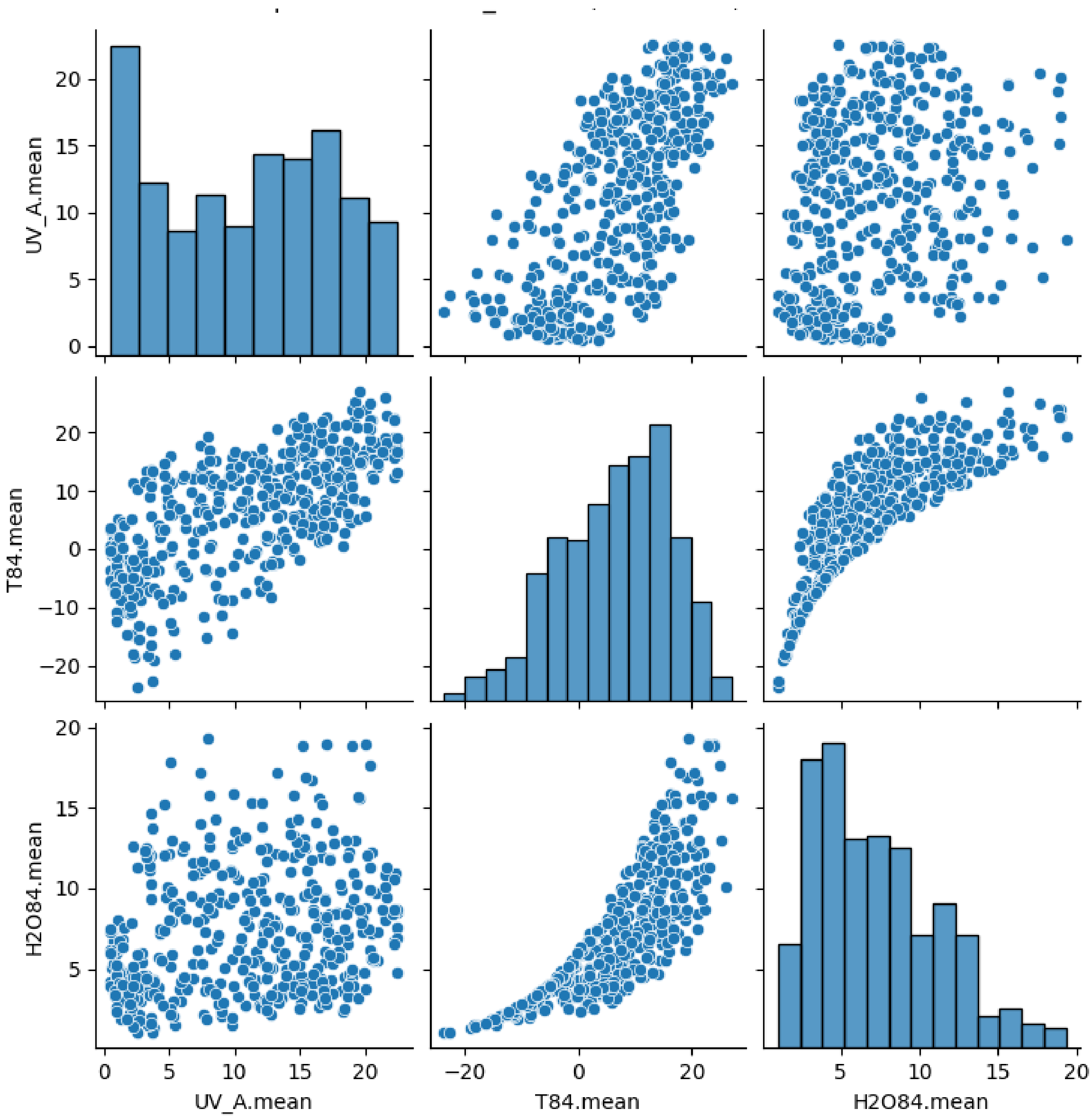
```
# task e
vars_to_plot = ["UV_A.mean", "T84.mean", "H2O84.mean"]
data_subset = df[vars_to_plot]

sns.pairplot(data_subset)
plt.suptitle("Scatterplot Matrix of UV_A.mean, T84.mean, H2O84.mean",
y=1.02)
plt.show()
```

```
   id        date    class4  partlybad  ...  UV_B.mean  UV_B.std   CS.mean
CS.std
0   0  2000-03-21        II      False  ...   0.115203  0.104295  0.000510
 0.000123
1   1  2000-03-23  nonevent      False  ...   0.301720  0.229672  0.000706
 0.000250
2   2  2000-04-07        Ia      False  ...   0.561251  0.451130  0.000851
 0.000244
3   3  2000-04-09        Ib      False  ...   0.716453  0.572409  0.002083
 0.000203
4   4  2000-04-14  nonevent      False  ...   0.146308  0.106017  0.002650
 0.000891

[5 rows x 104 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 450 entries, 0 to 449
Columns: 104 entries, id to CS.std
dtypes: bool(1), float64(100), int64(1), object(2)
memory usage: 362.7+ KB
None
               id  CO2168.mean  CO2168.std  ...    UV_B.std     CS.mean
CS.std
count  450.000000   450.000000  450.000000  ...  450.000000  450.000000
 450.000000
mean   224.500000   382.077392    3.352010  ...    0.379078    0.003083
 0.000692
std    130.048068    11.168050    3.448155  ...    0.285288    0.002246
 0.000678
min      0.000000   359.086782    0.120513  ...    0.002685    0.000343
 0.000023
25%    112.250000   373.872136    0.974968  ...    0.106026    0.001436
 0.000290
50%    224.500000   381.358306    2.228357  ...    0.360678    0.002548
 0.000525
75%    336.750000   389.397318    4.529128  ...    0.603543    0.004119
 0.000825
max    449.000000   414.863871   22.822280  ...    1.074115    0.013706
 0.006277

[8 rows x 101 columns]
Index(['date', 'class4', 'CO2168.mean', 'CO2168.std', 'CO2336.mean',
       'CO2336.std', 'CO242.mean', 'CO242.std', 'CO2504.mean', 'CO2504.std',
       ...
       'T672.mean', 'T672.std', 'T84.mean', 'T84.std', 'UV_A.mean',
'UV_A.std',
       'UV_B.mean', 'UV_B.std', 'CS.mean', 'CS.std'],
```

```
     dtype='object', length=102)
          T84.mean   UV_A.mean     CS.mean
count  450.000000  450.000000  450.000000
mean     6.439594   10.797330    0.003083
std      9.827282    6.626678    0.002246
min    -23.627710    0.438965    0.000343
25%     -0.733160    4.304971    0.001436
50%      7.661622   11.506937    0.002548
75%     14.118346   16.573188    0.004119
max     27.001804   22.500037    0.013706
the mean of t84.mean 6.439594405677285
the standard deviation of t84.mean 9.816356767300478
```








```python
import pandas as pd

path = './data/'

train_df = pd.read_csv(path+"train.csv")

most_common_class = train_df["class4"].mode()[0]

event_probability = (train_df["class4"] != "nonevent").mean()

print(f"Most common class: {most_common_class}")
print(f"Event probability: {event_probability:.4f}")

test_df = pd.read_csv(path+"test.csv")

test_df["class4"] = most_common_class
test_df["p"] = event_probability

submission = test_df[["id", "class4", "p"]]

submission.to_csv("dummy_submission.csv", index=False)
print("Dummy submission saved as dummy_submission.csv")
```

Problem2

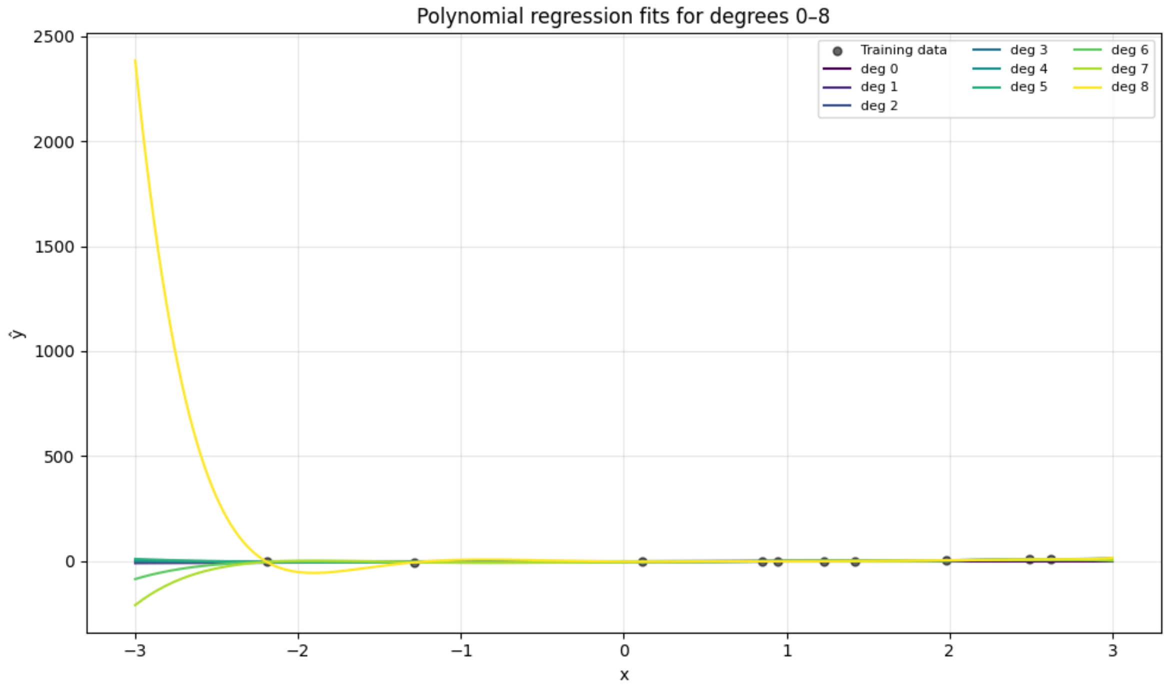
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score

RANDOM_STATE = 0

TRAIN_FILE = "./data/train_syn.csv"
VALID_FILE = "./data/valid_syn.csv"
TEST_FILE  = "./data/test_syn.csv"

train = pd.read_csv(TRAIN_FILE)
valid = pd.read_csv(VALID_FILE)
test  = pd.read_csv(TEST_FILE)

# Determine input and target column names automatically if only two columns
def infer_xy(df):
    if df.shape[1] == 2:
        cols = df.columns.tolist()
        return df[cols[0]].values.reshape(-1,1), df[cols[1]].values.ravel()
    else:
        # if more columns (e.g., id,date,...), user should replace this function
        raise ValueError("Expected CSV with exactly 2 columns (x,y) or modify this script to select appropriate columns")

X_train, y_train = infer_xy(train)
X_valid, y_valid = infer_xy(valid)
X_test,  y_test  = infer_xy(test)

# combined training+validation for CV and TestTRVA
X_trva = np.vstack([X_train, X_valid])
y_trva = np.concatenate([y_train, y_valid])

def mse_for_degree(deg, X_tr, y_tr, X_eval, y_eval):
    """Fit polynomial OLS on (X_tr,y_tr) and evaluate MSE on (X_eval,y_eval). Handles deg=0."""
    if deg == 0:
        # deg 0: constant model predicting the mean of y_tr
        y_pred = np.full_like(y_eval, fill_value=np.mean(y_tr), dtype=float)
        return mean_squared_error(y_eval, y_pred)
    # build polynomial features (no bias term; LinearRegression will learn intercept)
    poly = PolynomialFeatures(degree=deg, include_bias=False)
    X_tr_poly = poly.fit_transform(X_tr)
    X_eval_poly = poly.transform(X_eval)
    model = LinearRegression(fit_intercept=True)
    model.fit(X_tr_poly, y_tr)
```

```
    y_pred = model.predict(X_eval_poly)
    return mean_squared_error(y_eval, y_pred)

# function to compute 10-fold CV MSE (on combined train+valid)
def cv_mse_on_trva(deg, X_trva, y_trva, n_splits=10):
    if deg == 0:
        # CV of constant model: each fold's MSE = mean((y_val - mean(y_train_fold))^2)
        # Implement by manual KFold
        kf = KFold(n_splits=n_splits, shuffle=True, random_state=RANDOM_STATE)
        mses = []
        for train_idx, val_idx in kf.split(X_trva):
            y_tr_fold = y_trva[train_idx]
            y_val_fold = y_trva[val_idx]
            pred = np.full_like(y_val_fold, fill_value=np.mean(y_tr_fold), dtype=float)
            mses.append(mean_squared_error(y_val_fold, pred))
        return np.mean(mses)
    # use scikit's cross_val_score with neg_mean_squared_error
    poly = PolynomialFeatures(degree=deg, include_bias=False)
    X_trva_poly = poly.fit_transform(X_trva)
    model = LinearRegression(fit_intercept=True)
    # cross_val_score gives arrays of scores; for MSE use scoring='neg_mean_squared_error'
    scores = cross_val_score(model, X_trva_poly, y_trva,
                             scoring='neg_mean_squared_error',
                             cv=KFold(n_splits=n_splits, shuffle=True, random_state=RANDOM_STATE))
    return -np.mean(scores)  # return positive MSE

# Loop degrees 0..8 and compute columns
rows = []
for deg in range(0, 9):
    train_mse = mse_for_degree(deg, X_train, y_train, X_train, y_train)
    val_mse   = mse_for_degree(deg, X_train, y_train, X_valid, y_valid)
    test_mse  = mse_for_degree(deg, X_train, y_train, X_test, y_test)
    # TestTRVA: train on combined train+valid, test on test set
    test_trva_mse = mse_for_degree(deg, X_trva, y_trva, X_test, y_test)
    # CV: 10-fold on combined train+valid
    cv_mse = cv_mse_on_trva(deg, X_trva, y_trva, n_splits=10)
    rows.append({
        "Degree": deg,
        "Train": train_mse,
        "Validation": val_mse,
        "Test": test_mse,
        "TestTRVA": test_trva_mse,
        "CV": cv_mse
    })

df = pd.DataFrame(rows)
pd.set_option("display.float_format", lambda x: f"{x:.6f}")
print(df.to_string(index=False))

# save to CSV
```

```
# df.to_csv("degree_mse_table.csv", index=False)
# print("\nSaved table to degree_mse_table.csv")

import matplotlib.pyplot as plt

# Continuous x grid for smooth curve plotting
x_plot = np.linspace(-3, 3, 256).reshape(-1, 1)

plt.figure(figsize=(10, 6))

# Scatter training data for reference
plt.scatter(X_train, y_train, color="black", s=25, label="Training data",
alpha=0.6)

# Plot each polynomial fit
colors = plt.cm.viridis(np.linspace(0, 1, 9))  # one color per degree

for deg, color in zip(range(0, 9), colors):
    if deg == 0:
        # Constant model = mean of y_train
        y_plot = np.full_like(x_plot, fill_value=np.mean(y_train),
dtype=float)
    else:
        poly = PolynomialFeatures(degree=deg, include_bias=False)
        X_train_poly = poly.fit_transform(X_train)
        model = LinearRegression(fit_intercept=True)
        model.fit(X_train_poly, y_train)
        X_plot_poly = poly.transform(x_plot)
        y_plot = model.predict(X_plot_poly)
    plt.plot(x_plot, y_plot, color=color, lw=1.5, label=f"deg {deg}")

plt.xlabel("x")
plt.ylabel("ŷ")
plt.title("Polynomial regression fits for degrees 0–8")
plt.legend(ncol=3, fontsize=8)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()


import numpy as np
import pandas as pd
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor  # 5th model example
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

RANDOM_STATE = 0

# load data
train = pd.read_csv("./data/train_real.csv")
```

```python
test  = pd.read_csv("./data/test_real.csv")

# Separate features and target
y_train = train["Next_Tmax"].values
X_train = train.drop(columns=["Next_Tmax"]).values
y_test  = test["Next_Tmax"].values
X_test  = test.drop(columns=["Next_Tmax"]).values

# Standardise features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)

# Define regressors
regressors = {
    "Dummy": DummyRegressor(strategy="mean"),
    "OLS": LinearRegression(),
    "RF": RandomForestRegressor(
        n_estimators=200, max_depth=None, random_state=RANDOM_STATE, n_jobs=-1
    ),
    "SVR": SVR(kernel="rbf", C=10, epsilon=0.1),
    "KNN": KNeighborsRegressor(n_neighbors=5)  # 5th model (can be changed to Ridge, XGB, etc.)
}

# Function to compute RMSE and CV RMSE
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

def cross_val_rmse(model, X, y, n_splits=10):
    # cross_val_score returns negative MSE when scoring="neg_mean_squared_error"
    scores = cross_val_score(
        model, X, y,
        scoring="neg_mean_squared_error",
        cv=KFold(n_splits=n_splits, shuffle=True, random_state=RANDOM_STATE),
        n_jobs=-1
    )
    return np.sqrt(-scores.mean())

# train and evaluate
rows = []
for name, model in regressors.items():
    # choose scaled or unscaled depending on model
    Xtr = X_train_scaled if name in ["SVR", "KNN"] else X_train
    Xte = X_test_scaled if name in ["SVR", "KNN"] else X_test

    model.fit(Xtr, y_train)
    train_rmse = rmse(y_train, model.predict(Xtr))
    test_rmse  = rmse(y_test,  model.predict(Xte))

    Xcv = X_train_scaled if name in ["SVR", "KNN"] else X_train
    cv_rmse = cross_val_rmse(model, Xcv, y_train)
```

```python

    rows.append({"Regressor": name, "Train": train_rmse, "Test": test_rmse,
"CV": cv_rmse})

results = pd.DataFrame(rows)
pd.set_option("display.float_format", lambda x: f"{x:.4f}")
print(results.to_string(index=False))
```



Regressor	Train	Test	CV
Dummy	3.0893	2.9978	3.0922
OLS	1.3784	1.4838	1.4446
RF	0.5232	1.4732	1.4534
SVR	0.5671	1.5024	1.5695
KNN	1.4007	1.6944	1.7784

1. RF(Random Forest) is the best with the lowest RMSE
2. - Train vs Test

     RF and SVR overfits (Train much lower), OLS generalises most stably.
   - CV vs Test

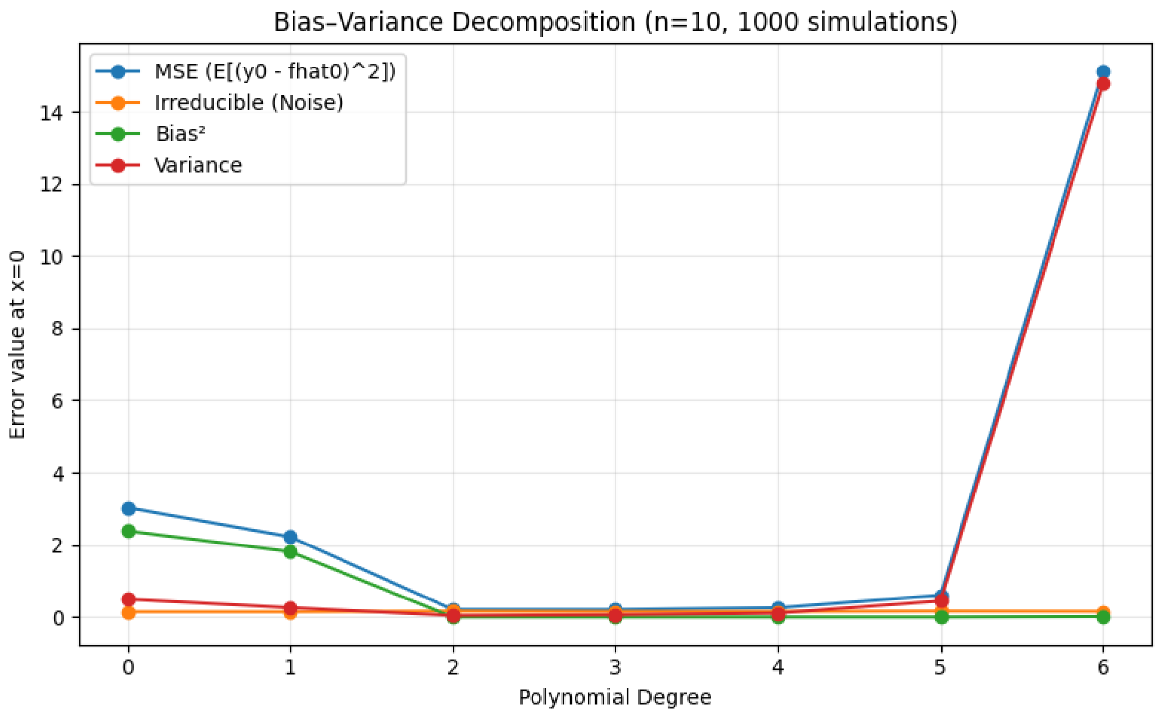
     Very close -> reliable estimate of test performance
3. Feature engineering, hyperparameter tuning, time-series CV

Problem3

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
```

```python
from sklearn.linear_model import LinearRegression

# params
np.random.seed(0)
sigma = 0.4  # noise
sigma2 = sigma ** 2
n_sim = 1000
n_train = 10  # train set size
degrees = range(0, 7)


def f(x):
    return -2 - x + 0.5 * x ** 2


f0 = f(0.0)  # f(0)

results = []

for deg in degrees:
    fhat0_list = []
    y0_list = []

    for _ in range(n_sim):
        # generate train set
        x_train = np.random.uniform(-3, 3, n_train).reshape(-1, 1)
        eps_train = np.random.normal(0, sigma, n_train)
        y_train = f(x_train.ravel()) + eps_train

        # fit model
        if deg == 0:
            fhat0 = np.mean(y_train)
        else:
            poly = PolynomialFeatures(degree=deg, include_bias=False)
            X_train = poly.fit_transform(x_train)
            model = LinearRegression()
            model.fit(X_train, y_train)
            fhat0 = model.predict(poly.transform([[0.0]]))[0]

        # generate a test sample (x=0, y0)
        y0 = f0 + np.random.normal(0, sigma)
        fhat0_list.append(fhat0)
        y0_list.append(y0)

    fhat0_arr = np.array(fhat0_list)
    y0_arr = np.array(y0_list)

    # compute
    irreducible = np.mean((y0_arr - f0) ** 2)
    bias = np.mean(fhat0_arr) - f0
    bias_sq = bias ** 2
    variance = np.mean((fhat0_arr - np.mean(fhat0_arr)) ** 2)
    total = irreducible + bias_sq + variance
    mse = np.mean((y0_arr - fhat0_arr) ** 2)

```

```
    results.append({
        "Degree": deg,
        "Irreducible": irreducible,
        "BiasSq": bias_sq,
        "Variance": variance,
        "Total (sum)": total,
        "MSE": mse
    })

df = pd.DataFrame(results)
print(df.round(6))

plt.figure(figsize=(8, 5))
plt.plot(df["Degree"], df["MSE"], 'o-', label="MSE (E[(y0 - fhat0)^2])")
plt.plot(df["Degree"], df["Irreducible"], 'o-', label="Irreducible (Noise)")
plt.plot(df["Degree"], df["BiasSq"], 'o-', label="Bias²")
plt.plot(df["Degree"], df["Variance"], 'o-', label="Variance")
plt.xlabel("Polynomial Degree")
plt.ylabel("Error value at x=0")
plt.title("Bias-Variance Decomposition (n=10, 1000 simulations)")
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

```



Less flexible -> simple model

more flexible -> complex model

Term	Meaning
Training error	Error measured on training data (fit data).

Term	Meaning
Testing error	Error measured on unseen data.
(Squared) Bias	How far the model's expected predictions are from the true function (systematic error).
Variance	How much the model's predictions vary with different training samples (sensitivity to data).
Irreducible error	Noise inherent in the data; cannot be reduced by any model.

Term	Behaviour	Explanation
Training error	**Decreases monotonically**	Flexible models can always fit the training data better, even noise.
Testing error	**Decreases first, then increases (U-shaped curve)**	Initially, more flexibility reduces bias → better test performance; later, variance dominates → overfitting.
Bias²	**Decreases monotonically**	Simple models underfit (high bias); more flexible models approximate the true function better.
Variance	**Increases monotonically**	Flexible models are sensitive to data fluctuations; small changes in training set → large prediction changes.
Irreducible error	**Constant (flat line)**	Comes from random noise in Y (measurement or inherent randomness). No model can remove it.

Do the terms behave as expected? Does Total ≈ MSE?

For degrees lower than the true model (0 and 1), Bias² is large, variance modest → high MSE.

At degree 2 (the true quadratic), Bias² nearly zero, variance near zero → MSE ≈ irreducible (~0.16).

For moderate degrees (3–5), both bias and variance are small; total remains low and near irreducible.

Numerically, **Total = Irreducible + Bias² + Variance** matches the Monte Carlo estimate of **MSE**, confirming the decomposition.

Problem4

Taska

$$E[L_{test}] = \frac{1}{m}\sum_{i=1}^{m} E[(\overline{y}_i - \hat{\beta}^T \overline{x}_i)^2] \tag{1}$$

Cause each pair `(xi,yi)` was drawn at random with replacement, we can consider

$$(x_1, y_1)\text{~}(x_2, y_2)\text{~}(x_m, y_m) \tag{2}$$

So

$$
\begin{aligned}
E[L_{test}] &= \frac{1}{m}\sum_{i=1}^{m} E[(\overline{y}_i - \hat{\beta}^T\overline{x}_i)^2] \\
&= \frac{1}{m}\cdot m \cdot E[(\overline{y}_1 - \hat{\beta}^T\overline{x}_1)^2] \\
&= E[(\overline{y}_1 - \hat{\beta}^T\overline{x}_1)^2]
\end{aligned} \tag{3}
$$

Taskb

$$
GenError = E[(y - \hat{\beta}^T x)^2] = E[L_{test}] \tag{4}
$$

Cause the expectation here is over both the randomness in training set and the test point, then `L_test` is unbiased for the generalization error.

Taskc

For train set, we choose `β` to minimize `L_train`

But for a new point out of train set, we can't choose `β` to minimize error

So

$$
E[L_{train}] <= E[L_{test}] \tag{5}
$$

Taskd

Training loss is optimistic (usually smaller) compared to test loss.

Test loss gives a better estimate of generalization error.

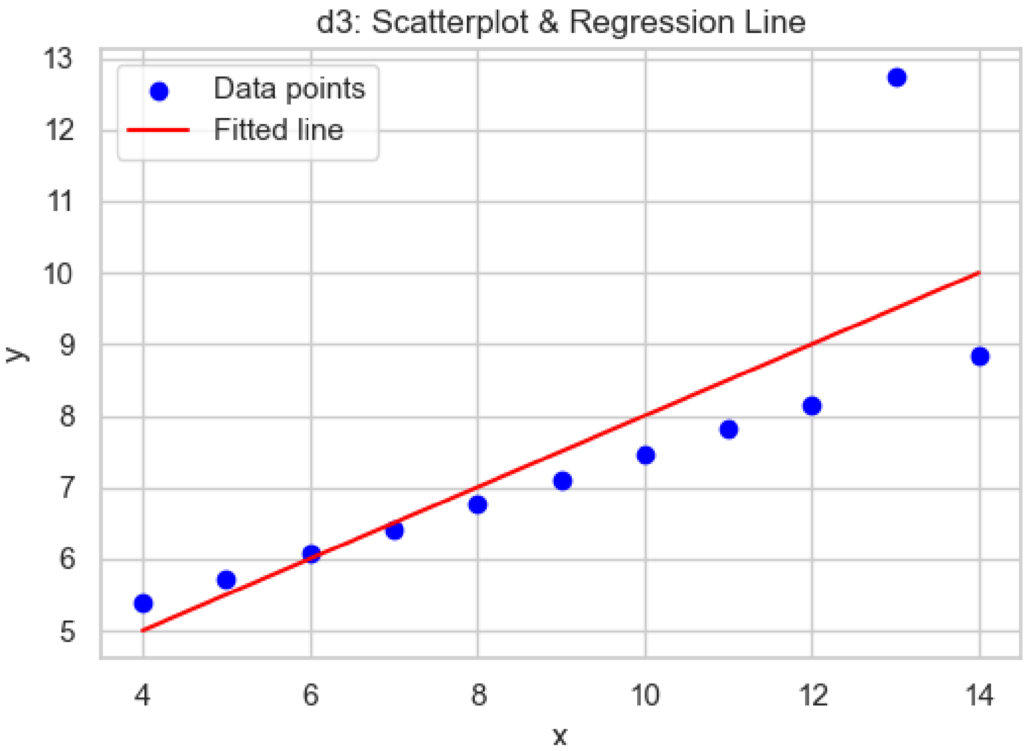
In machine learning: minimizing training loss alone can lead to overfitting. True performance is measured on independent test data.

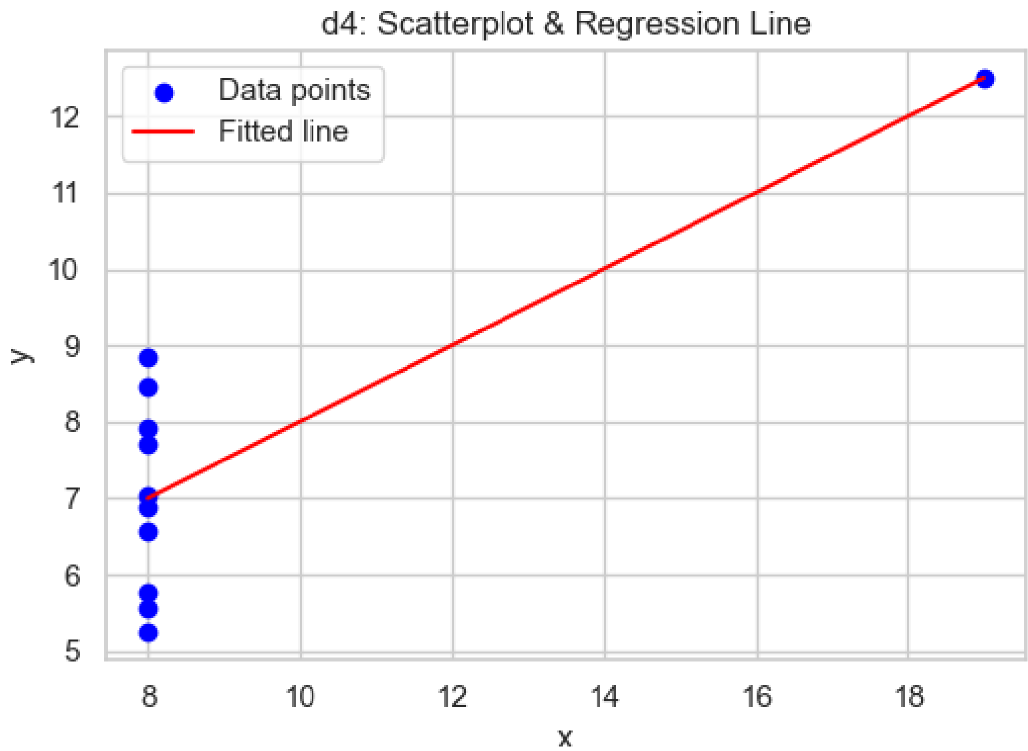
Problem5

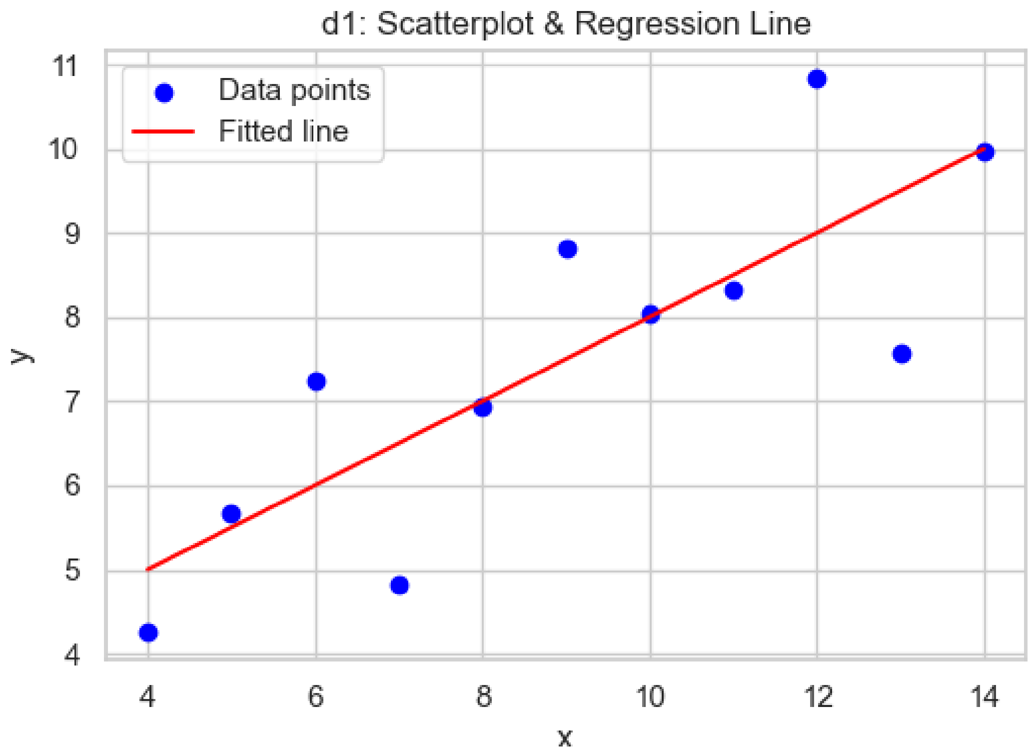
```python
import pandas as pd
import statsmodels.api as sm

d1 = pd.read_csv('./data/d1.csv')
d2 = pd.read_csv('./data/d2.csv')
d3 = pd.read_csv('./data/d3.csv')
d4 = pd.read_csv('./data/d4.csv')

def fit_ols(df):
    X = sm.add_constant(df['x'])  # adds intercept term w0
    y = df['y']
    model = sm.OLS(y, X).fit()
    return model

models = [fit_ols(d) for d in [d1, d2, d3, d4]]

# for i, model in enumerate(models, 1):
#     print(f"Dataset d{i}:")
#     print(model.summary())
#     print("\n")

for i, model in enumerate(models, 1):
```

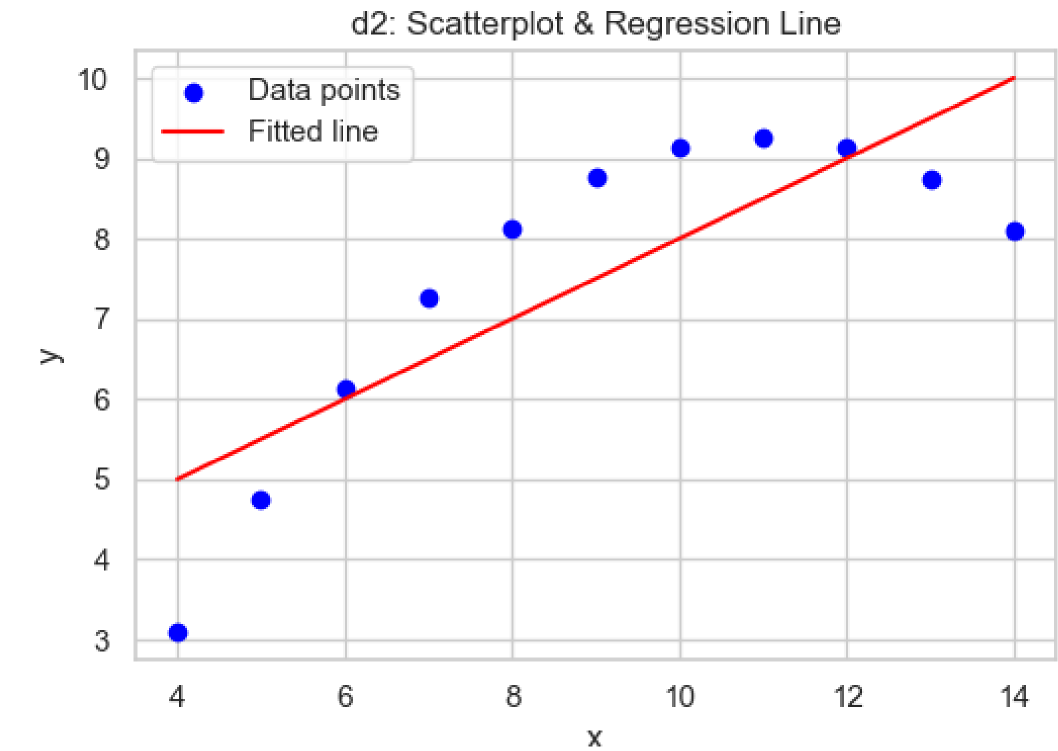
```
    print(f"Dataset d{i}:")
    print(f"Intercept (w0): {model.params[0]:.4f}, SE={model.bse[0]:.4f}, p=
{model.pvalues[0]:.4f}")
    print(f"Slope (w1): {model.params[1]:.4f}, SE={model.bse[1]:.4f}, p=
{model.pvalues[1]:.4f}")
    print(f"R-squared: {model.rsquared:.4f}")
    print()

import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="whitegrid")


def plot_regression(df, model, dataset_name):
    plt.figure(figsize=(6, 4))

    # Scatter plot of the data
    plt.scatter(df['x'], df['y'], color='blue', label='Data points')

    # Regression line
    X_range = pd.DataFrame({'x': sorted(df['x'])})
    X_range = sm.add_constant(X_range)  # add intercept
    y_pred = model.predict(X_range)
    plt.plot(sorted(df['x']), y_pred, color='red', label='Fitted line')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'{dataset_name}: Scatterplot & Regression Line')
    plt.legend()
    plt.show()

datasets = [d1, d2, d3, d4]
for i, (df, model) in enumerate(zip(datasets, models), 1):
    plot_regression(df, model, f'd{i}')


def diagnostic_plot(model, dataset_name):
    residuals = model.resid
    fitted = model.fittedvalues

    plt.figure(figsize=(6, 4))
    plt.scatter(fitted, residuals)
    plt.axhline(0, color='red', linestyle='--')
    plt.xlabel('Fitted values')
    plt.ylabel('Residuals')
    plt.title(f'{dataset_name}: Residuals vs Fitted')
    plt.show()


# Apply to each dataset
for i, model in enumerate(models, 1):
    diagnostic_plot(model, f'd{i}')

```

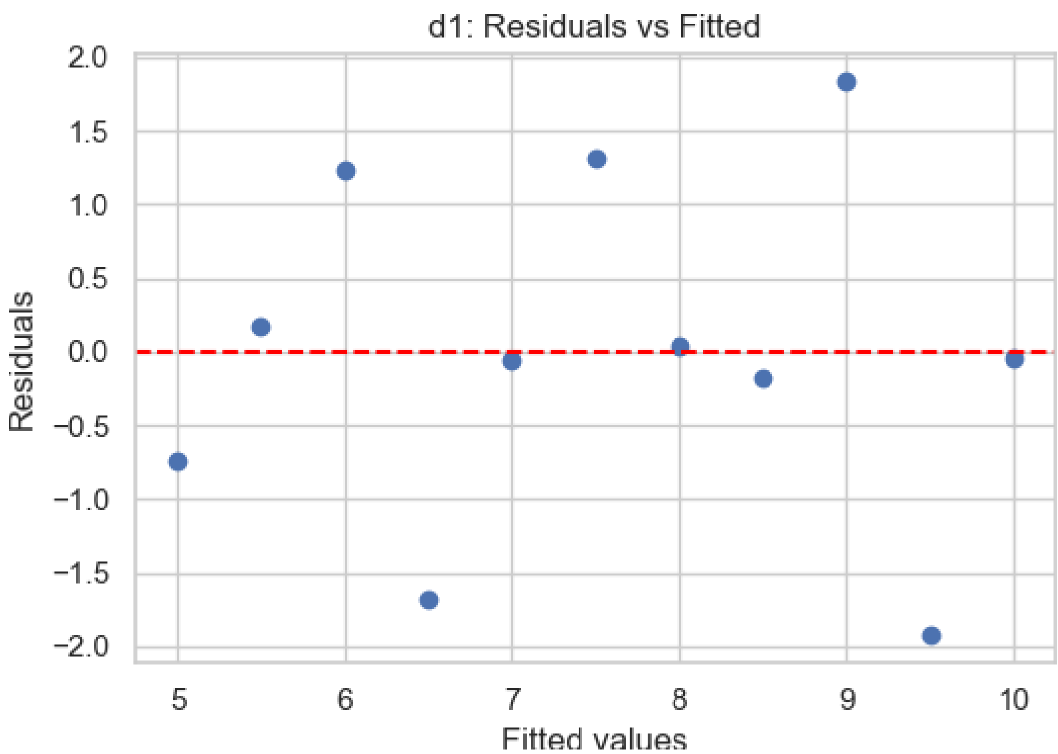
```
Dataset d1:
Intercept (w0): 3.0001, SE=1.1247, p=0.0257
Slope (w1): 0.5001, SE=0.1179, p=0.0022
R-squared: 0.6665

Dataset d2:
Intercept (w0): 3.0009, SE=1.1253, p=0.0258
Slope (w1): 0.5000, SE=0.1180, p=0.0022
R-squared: 0.6662

Dataset d3:
Intercept (w0): 3.0025, SE=1.1245, p=0.0256
Slope (w1): 0.4997, SE=0.1179, p=0.0022
R-squared: 0.6663

Dataset d4:
Intercept (w0): 3.0017, SE=1.1239, p=0.0256
Slope (w1): 0.4999, SE=0.1178, p=0.0022
R-squared: 0.6667
```

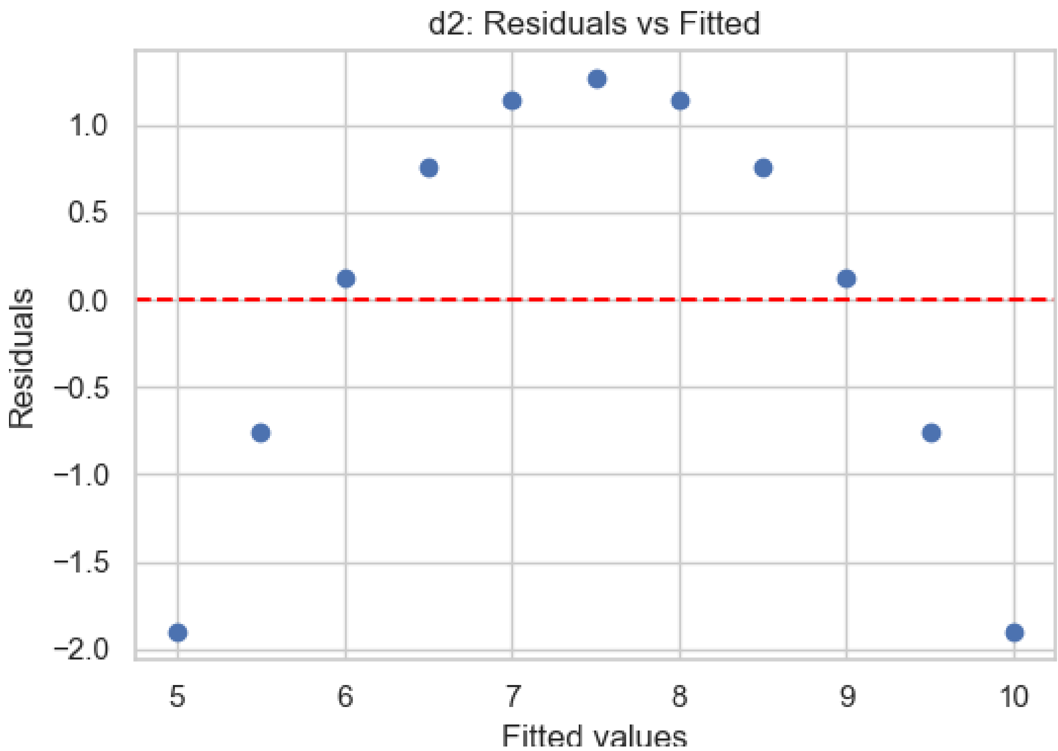


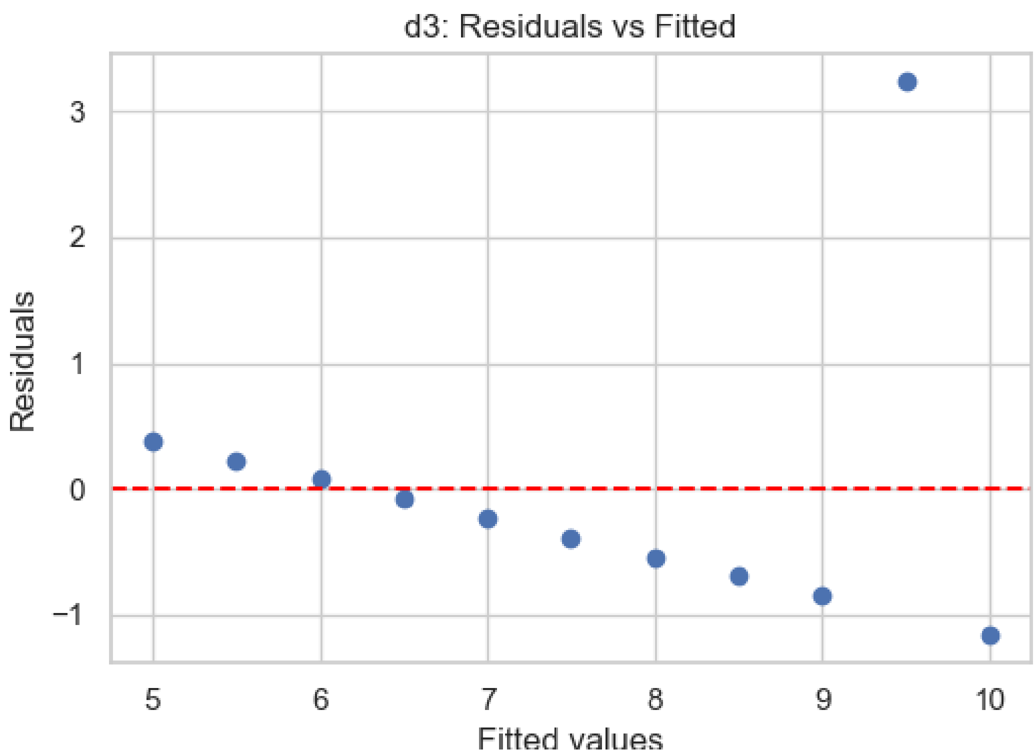

d4: Scatterplot & Regression Line
Data points
Fitted line
y
x
5
6
7
8
9
10
11
12
8
10
12
14
16
18

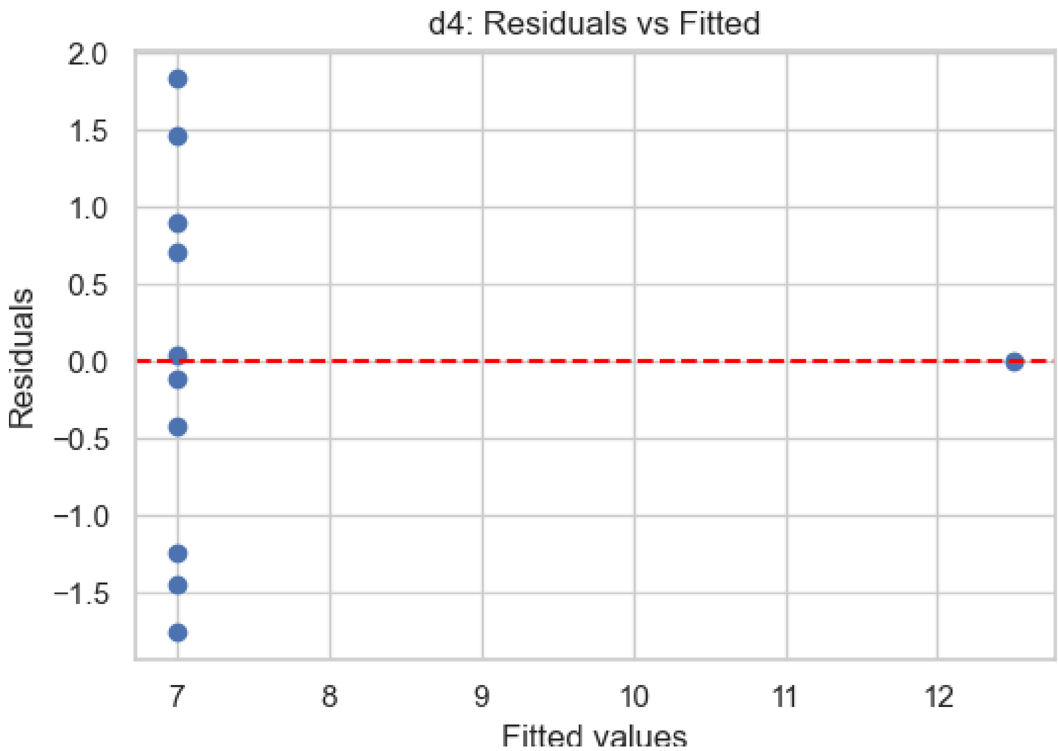

d1: Scatterplot & Regression Line
Data points
Fitted line
y
x
4
5
6
7
8
9
10
11
4
6
8
10
12
14


d2: Scatterplot & Regression Line
Data points
Fitted line
10
9
8
7
6
5
4
3
y
4
6
8
10
12
14
x


d1: Residuals vs Fitted
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Residuals
5
6
7
8
9
10
Fitted values


d2: Residuals vs Fitted
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Residuals
5
6
7
8
9
10
Fitted values


d3: Residuals vs Fitted
3
2
1
0
−1
Residuals
5
6
7
8
9
10
Fitted values



Taskb

Only a small number of points fall on the line, resulting in a very poor fit.

Taskc

- non-linearity d2 d4
- Outliers d4
- High leverage points d3

Problem6

```
import numpy as np
import statsmodels.api as sm
import pandas as pd

# Load d2.csv
d2 = pd.read_csv('./data/d2.csv')

X = sm.add_constant(d2['x'])
y = d2['y']

# Number of bootstrap samples
B = 1000
w0_boot = []
w1_boot = []

n = len(d2)

for _ in range(B):
    # sample indices with replacement
    sample_indices = np.random.choice(n, n, replace=True)
    X_sample = X.iloc[sample_indices]
```

```python
    y_sample = y.iloc[sample_indices]

    # fit OLS
    model = sm.OLS(y_sample, X_sample).fit()

    w0_boot.append(model.params[0])
    w1_boot.append(model.params[1])

# Compute bootstrap standard errors
se_w0_boot = np.std(w0_boot, ddof=1)
se_w1_boot = np.std(w1_boot, ddof=1)

print(f"Bootstrap SE for w0: {se_w0_boot:.4f}")
print(f"Bootstrap SE for w1: {se_w1_boot:.4f}")

```

```
Bootstrap SE for w0: 1.6232
Bootstrap SE for w1: 0.1725
```

taskb

The bootstrap algorithm estimates the standard errors of the regression coefficients by simulating many repeated samples from the data.

1. **Resample the data:**
    - From the original dataset, draw a new sample with replacement of the same size.
2. **Fit the regression model to the resampled data:**
    - Compute the estimates of the intercept (`w0`) and slope (`w1`) for this bootstrap sample.
3. **Repeat many times:**
    - Perform steps 1–2 for a large number of bootstrap samples (e.g., 1000).
    - This gives a distribution of estimated coefficients for `w0` and `w1`.
4. **Compute standard errors:**
    - The standard deviation of these bootstrap estimates approximates the standard error of each coefficient.

taskc

$$P(not\ chosen) = 1 - \frac{1}{n} \tag{6}$$

$$P(never\ chosen) = (1 - \frac{1}{n})^n \tag{7}$$

$$lim_{n \to \infty}(1 - \frac{1}{n})^n = \frac{1}{e} \approx 0.368 \tag{8}$$

Problem7

Task a

During lectures 1–4 and this exercise set, I learned the fundamentals of linear regression, including how to model relationships between variables, interpret regression coefficients, and evaluate model performance using statistical metrics such as R^2 and p-values. I also gained a better understanding of the assumptions behind linear models, including linearity, independence, homoscedasticity, and normality of residuals. Through the practical exercises, I learned how to use Python and statsmodels to fit regression models, visualize data, and interpret output tables.

One challenging aspect was understanding model diagnostics and how to identify problems such as nonlinearity or multicollinearity. I also realized that interpreting regression coefficients can be tricky when variables are correlated or transformed.

Task b

7-8 hours