

PYTHON SCRIPTS

Linear Regression

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVR

from sklearn.neighbors import KNeighborsRegressor

from sklearn.neural_network import MLPRegressor

from keras.models import Sequential

from keras.layers import Dense

!pip install matplotlib

from google.colab import files

uploaded = files.upload()

import pandas as pd

import numpy as np

from sklearn.model_selection import KFold, cross_val_score

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make_pipeline
```

```
from sklearn.metrics import make_scorer, mean_squared_error, mean_absolute_error, r2_score
```

```
csv_file = 'points.csv'
```

```
random_state = 42
```

```
n_splits = 5
```

```
df = pd.read_csv(csv_file)
```

```
features = ["temperature", "density", "refractive index"]
```

```
X = df[features]
```

```
y_abw = df["ABW"]
```

```
y_brix = df["brix"]
```

```
mse_scorer = make_scorer(mean_squared_error)
```

```
mae_scorer = make_scorer(mean_absolute_error)
```

```
r2_scorer = make_scorer(r2_score)
```

```
kf = KFold(n_splits=n_splits, shuffle=True, random_state=random_state)
```

```
def cross_validate_target(X, y, label):
```

```
    print(f"\n--- {label} (Unscaled) ---")
```

```
    model = LinearRegression()
```

```

mse_scores = cross_val_score(model, X, y, cv=kf, scoring=mse_scorer)
mae_scores = cross_val_score(model, X, y, cv=kf, scoring=mae_scorer)
r2_scores = cross_val_score(model, X, y, cv=kf, scoring=r2_scorer)
print(f"MSE: {np.mean(mse_scores):.4f}")
print(f"MAE: {np.mean(mae_scores):.4f}")
print(f"R2: {np.mean(r2_scores):.4f}")

print(f"--- {label} (Scaled) ---")
pipeline = make_pipeline(StandardScaler(), LinearRegression())
mse_scores = cross_val_score(pipeline, X, y, cv=kf, scoring=mse_scorer)
mae_scores = cross_val_score(pipeline, X, y, cv=kf, scoring=mae_scorer)
r2_scores = cross_val_score(pipeline, X, y, cv=kf, scoring=r2_scorer)
print(f"MSE: {np.mean(mse_scores):.4f}")
print(f"MAE: {np.mean(mae_scores):.4f}")
print(f"R2: {np.mean(r2_scores):.4f}")

```

```

cross_validate_target(X, y_abw, "ABW")
cross_validate_target(X, y_brix, "Brix")

```

```

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

```

```

def plot_pred_vs_actual(y_true, y_pred, title):

```

```

plt.figure(figsize=(5,5))
plt.scatter(y_true, y_pred, alpha=0.7)
min_val = min(y_true.min(), y_pred.min())
max_val = max(y_true.max(), y_pred.max())
plt.plot([min_val, max_val], [min_val, max_val], 'r--', label='x = y')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title(title)
plt.legend()
plt.tight_layout()
plt.show()

```

```

X_train, X_test, y_abw_train, y_abw_test = train_test_split(
    X, y_abw, test_size=0.2, random_state=random_state
)

```

```

X_train2, X_test2, y_brix_train, y_brix_test = train_test_split(
    X, y_brix, test_size=0.2, random_state=random_state
)

```

Unscaled models

```

model_abw = LinearRegression().fit(X_train, y_abw_train)
y_abw_pred = model_abw.predict(X_test)
model_brix = LinearRegression().fit(X_train2, y_brix_train)
y_brix_pred = model_brix.predict(X_test2)

```

```
# Scaled models
```

```
scaler_abw = StandardScaler().fit(X_train)
```

```
X_train_scaled = scaler_abw.transform(X_train)
```

```
X_test_scaled = scaler_abw.transform(X_test)
```

```
model_abw_scaled = LinearRegression().fit(X_train_scaled, y_abw_train)
```

```
y_abw_pred_scaled = model_abw_scaled.predict(X_test_scaled)
```

```
scaler_brix = StandardScaler().fit(X_train2)
```

```
X_train2_scaled = scaler_brix.transform(X_train2)
```

```
X_test2_scaled = scaler_brix.transform(X_test2)
```

```
model_brix_scaled = LinearRegression().fit(X_train2_scaled, y_brix_train)
```

```
y_brix_pred_scaled = model_brix_scaled.predict(X_test2_scaled)
```

```
# Plot for ABW
```

```
plot_pred_vs_actual(y_abw_test, y_abw_pred, "ABW (Unscaled)")
```

```
plot_pred_vs_actual(y_abw_test, y_abw_pred_scaled, "ABW (Scaled)")
```

```
# Plot for Brix
```

```
plot_pred_vs_actual(y_brix_test, y_brix_pred, "Brix (Unscaled)")
```

```
plot_pred_vs_actual(y_brix_test, y_brix_pred_scaled, "Brix (Scaled)")
```

SVM regression

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```

import seaborn as sns

from sklearn.svm import SVR

from sklearn.model_selection import train_test_split, cross_val_score, KFold

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import warnings

warnings.filterwarnings('ignore')


np.random.seed(42)


data = pd.read_csv("points.csv", header=0)


print("Data loaded successfully!")

print(f"Data shape: {data.shape}")

print("\nFirst few rows:")

print(data.head())

print("\nColumn names:")

print(data.columns.tolist())

print("\nData info:")

print(data.info())


X = data[['temperature', 'density', 'refractive index']].values

y_abw = data['ABW'].values

```

```
y_brix = data['brix'].values
```

```
print(f"\nFeatures shape: {X.shape}")
```

```
print(f"ABW target shape: {y_abw.shape}")
```

```
print(f"BRIX target shape: {y_brix.shape}")
```

```
def evaluate_model(y_true, y_pred, model_name, target_name, data_type):
```

```
    mse = mean_squared_error(y_true, y_pred)
```

```
    r2 = r2_score(y_true, y_pred)
```

```
    mae = mean_absolute_error(y_true, y_pred)
```

```
    print(f"\n{model_name} - {target_name} ({data_type}):")
```

```
    print(f"MSE: {mse:.4f}")
```

```
    print(f"R2: {r2:.4f}")
```

```
    print(f"MAE: {mae:.4f}")
```

```
    return {'MSE': mse, 'R2': r2, 'MAE': mae}
```

```
def plot_predictions(y_true_train, y_pred_train, y_true_test, y_pred_test, target_name,
scaled_status):
```

```
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
```

```
    ax1.scatter(y_true_train, y_pred_train, alpha=0.6, color='blue')
```

```

ax1.plot([y_true_train.min(), y_true_train.max()],
         [y_true_train.min(), y_true_train.max()], 'r--', lw=2)
ax1.set_xlabel(f'Actual {target_name}')
ax1.set_ylabel(f'Predicted {target_name}')
ax1.set_title(f'SVM - {target_name} Training ({scaled_status})')
ax1.grid(True, alpha=0.3)

r2_train = r2_score(y_true_train, y_pred_train)
ax1.text(0.05, 0.95, f'R2 = {r2_train:.3f}', transform=ax1.transAxes,
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

ax2.scatter(y_true_test, y_pred_test, alpha=0.6, color='green')
ax2.plot([y_true_test.min(), y_true_test.max()],
         [y_true_test.min(), y_true_test.max()], 'r--', lw=2)
ax2.set_xlabel(f'Actual {target_name}')
ax2.set_ylabel(f'Predicted {target_name}')
ax2.set_title(f'SVM - {target_name} Testing ({scaled_status})')
ax2.grid(True, alpha=0.3)

r2_test = r2_score(y_true_test, y_pred_test)
ax2.text(0.05, 0.95, f'R2 = {r2_test:.3f}', transform=ax2.transAxes,
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

```

```
plt.tight_layout()
```

```
plt.show()
```

```
# cross-validation
```

```
def cross_validation_analysis(X, y, model, target_name, scaled_status, cv_folds=5):
```

```
    print(f"\n{cv_folds}-Fold Cross-Validation - {target_name} ({scaled_status}):")
```

```
    cv_mse = -cross_val_score(model, X, y, cv=cv_folds, scoring='neg_mean_squared_error')
```

```
    cv_r2 = cross_val_score(model, X, y, cv=cv_folds, scoring='r2')
```

```
    cv_mae = -cross_val_score(model, X, y, cv=cv_folds, scoring='neg_mean_absolute_error')
```

```
    print(f"MSE: {cv_mse.mean():.4f} (+/- {cv_mse.std() * 2:.4f})")
```

```
    print(f"R2: {cv_r2.mean():.4f} (+/- {cv_r2.std() * 2:.4f})")
```

```
    print(f"MAE: {cv_mae.mean():.4f} (+/- {cv_mae.std() * 2:.4f})")
```

```
    return {
```

```
        'MSE_mean': cv_mse.mean(), 'MSE_std': cv_mse.std(),
```

```
        'R2_mean': cv_r2.mean(), 'R2_std': cv_r2.std(),
```

```
        'MAE_mean': cv_mae.mean(), 'MAE_std': cv_mae.std()
```

```
    }
```

```
targets = [('ABW', y_abw), ('brix', y_brix)]
```

```
results = {}
```

```
for target_name, y in targets:
```

```
    print(f"\n{' '*60}")
```

```
    print(f"ANALYSIS FOR {target_name.upper()}")
```

```
    print(f"{' '*60}")
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
results[target_name] = {}
```

```
# UNSCALED data
```

```
print(f"\n{' '*40}")
```

```
print("SVM REGRESSION - UNSCALED DATA")
```

```
print(f"{' '*40}")
```

```
svm_unscaled = SVR(kernel='rbf', C=1.0, gamma='scale')
```

```
svm_unscaled.fit(X_train, y_train)
```

```
y_pred_train_unscaled = svm_unscaled.predict(X_train)
```

```
y_pred_test_unscaled = svm_unscaled.predict(X_test)
```

```
train_metrics_unscaled = evaluate_model(y_train, y_pred_train_unscaled, "SVM",  
target_name, "Training - Unscaled")
```

```
test_metrics_unscaled = evaluate_model(y_test, y_pred_test_unscaled, "SVM", target_name,  
"Testing - Unscaled")
```

```
cv_metrics_unscaled = cross_validation_analysis(X, y, svm_unscaled, target_name,
"Unscaled", 5)
```

```
plot_predictions(y_train, y_pred_train_unscaled, y_test, y_pred_test_unscaled, target_name,
"Unscaled Data")
```

```
results[target_name]['unscaled'] = {
    'train_metrics': train_metrics_unscaled,
    'test_metrics': test_metrics_unscaled,
    'cv_metrics': cv_metrics_unscaled
}
```

```
# SCALED data
print(f"\n{'-'*40}")
print("SVM REGRESSION - SCALED DATA")
print(f"{'-'*40}")
```

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

```

svm_scaled = SVR(kernel='rbf', C=1.0, gamma='scale')
svm_scaled.fit(X_train_scaled, y_train)

y_pred_train_scaled = svm_scaled.predict(X_train_scaled)
y_pred_test_scaled = svm_scaled.predict(X_test_scaled)

train_metrics_scaled = evaluate_model(y_train, y_pred_train_scaled, "SVM", target_name,
"Training - Scaled")

test_metrics_scaled = evaluate_model(y_test, y_pred_test_scaled, "SVM", target_name,
"Testing - Scaled")

cv_metrics_scaled = cross_validation_analysis(X_full_scaled, y, svm_scaled, target_name,
"Scaled", 5)

plot_predictions(y_train, y_pred_train_scaled, y_test, y_pred_test_scaled, target_name,
"Scaled Data")

results[target_name]['scaled'] = {
    'train_metrics': train_metrics_scaled,
    'test_metrics': test_metrics_scaled,
    'cv_metrics': cv_metrics_scaled
}

```

```

targets = list(results.keys())

metrics = ['MSE', 'R2', 'MAE']

fig, axes = plt.subplots(len(targets), len(metrics), figsize=(15, 10))

if len(targets) == 1:
    axes = axes.reshape(1, -1)

for i, target in enumerate(targets):
    for j, metric in enumerate(metrics):
        ax = axes[i, j]

        unscaled_value = results[target]['unscaled']['test_metrics'][metric]
        scaled_value = results[target]['scaled']['test_metrics'][metric]

        bars = ax.bar(['Unscaled', 'Scaled'], [unscaled_value, scaled_value],
                       color=['lightcoral', 'lightblue'])

        ax.set_title(f'{target} - {metric}')
        ax.set_ylabel(metric)

        for bar, value in zip(bars, [unscaled_value, scaled_value]):
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height,
                    f'{value:.3f}', ha='center', va='bottom')

```

```
plt.tight_layout()
```

```
plt.show()
```

```
print(f"\n{' '*80}")
```

```
print("COMPREHENSIVE RESULTS SUMMARY")
```

```
print(f"{' '*80}")
```

```
for target in results:
```

```
    print(f"\n{target.upper()} PREDICTION RESULTS:")
```

```
    print(f"{' '*50}")
```

```
    for scaling_type in ['unscaled', 'scaled']:
```

```
        print(f"\n{scaling_type.upper()} DATA:")
```

```
        test_metrics = results[target][scaling_type]['test_metrics']
```

```
        print(f" Test MSE: {test_metrics['MSE']:.4f}")
```

```
        print(f" Test R2: {test_metrics['R2']:.4f}")
```

```
        print(f" Test MAE: {test_metrics['MAE']:.4f}")
```

```
        cv_metrics = results[target][scaling_type]['cv_metrics']
```

```
        print(f" CV R2 (mean ± std): {cv_metrics['R2_mean']:.4f} ± {cv_metrics['R2_std']:.4f}")
```

```
import pandas as pd
```

```

import numpy as np

import matplotlib.pyplot as plt

from sklearn.svm import SVR

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import warnings

warnings.filterwarnings('ignore')


np.random.seed(42)


data = pd.read_csv("points.csv", header=0)

X = data[['temperature', 'density', 'refractive index']].values

y_abw = data['ABW'].values

y_brix = data['brix'].values


# Test different epsilon values

epsilon_values = [0.01, 0.05, 0.1, 0.2, 0.5, 1.0]


def test_epsilon_values(X, y, target_name, epsilon_values):

    """Test different epsilon values and compare performance"""


    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

```

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_full_scaled = scaler.fit_transform(X)

results = []

print(f"\n{' '*60}")
print(f"EPSILON TESTING FOR {target_name.upper()}")
print(f"{' '*60}")

for eps in epsilon_values:

    svm = SVR(kernel='rbf', C=1.0, gamma='scale', epsilon=eps)
    svm.fit(X_train_scaled, y_train)

    y_pred_train = svm.predict(X_train_scaled)
    y_pred_test = svm.predict(X_test_scaled)

    train_r2 = r2_score(y_train, y_pred_train)
    test_r2 = r2_score(y_test, y_pred_test)
    train_mse = mean_squared_error(y_train, y_pred_train)

```

```
test_mse = mean_squared_error(y_test, y_pred_test)
```

```
cv_scores = cross_val_score(svm, X_full_scaled, y, cv=5, scoring='r2')
```

```
cv_r2_mean = cv_scores.mean()
```

```
n_support_vectors = len(svm.support_)
```

```
results.append({  
    'epsilon': eps,  
    'train_r2': train_r2,  
    'test_r2': test_r2,  
    'train_mse': train_mse,  
    'test_mse': test_mse,  
    'cv_r2_mean': cv_r2_mean,  
    'n_support_vectors': n_support_vectors  
})
```

```
print(f"\nEpsilon = {eps:4.2f}:")
```

```
print(f" Train R2: {train_r2:.4f} | Test R2: {test_r2:.4f}")
```

```
print(f" Train MSE: {train_mse:.4f} | Test MSE: {test_mse:.4f}")
```

```
print(f" CV R2 (5-fold): {cv_r2_mean:.4f}")
```

```
print(f" Support Vectors: {n_support_vectors}")
```

```
return results
```

```
abw_results = test_epsilon_values(X, y_abw, 'ABW', epsilon_values)
```

```
brix_results = test_epsilon_values(X, y_brix, 'BRIX', epsilon_values)
```

```
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
```

```
axes[0,0].plot([r['epsilon'] for r in abw_results], [r['train_r2'] for r in abw_results], 'b-o', label='Train  
R2)
```

```
axes[0,0].plot([r['epsilon'] for r in abw_results], [r['test_r2'] for r in abw_results], 'r-o', label='Test  
R2)
```

```
axes[0,0].plot([r['epsilon'] for r in abw_results], [r['cv_r2_mean'] for r in abw_results], 'g-o',  
label='CV R2)
```

```
axes[0,0].set_xlabel('Epsilon')
```

```
axes[0,0].set_ylabel('R2 Score')
```

```
axes[0,0].set_title('ABW - R2 vs Epsilon')
```

```
axes[0,0].legend()
```

```
axes[0,0].grid(True, alpha=0.3)
```

```
axes[0,1].plot([r['epsilon'] for r in abw_results], [r['n_support_vectors'] for r in abw_results],  
'purple', marker='o')
```

```
axes[0,1].set_xlabel('Epsilon')
```

```
axes[0,1].set_ylabel('Number of Support Vectors')
```

```
axes[0,1].set_title('ABW - Model Complexity vs Epsilon')
```

```

axes[0,1].grid(True, alpha=0.3)

axes[1,0].plot([r['epsilon'] for r in brix_results], [r['train_r2'] for r in brix_results], 'b-o', label='Train
R²')
axes[1,0].plot([r['epsilon'] for r in brix_results], [r['test_r2'] for r in brix_results], 'r-o', label='Test R²')
axes[1,0].plot([r['epsilon'] for r in brix_results], [r['cv_r2_mean'] for r in brix_results], 'g-o',
label='CV R²')
axes[1,0].set_xlabel('Epsilon')
axes[1,0].set_ylabel('R² Score')
axes[1,0].set_title('BRIX - R² vs Epsilon')
axes[1,0].legend()
axes[1,0].grid(True, alpha=0.3)

axes[1,1].plot([r['epsilon'] for r in brix_results], [r['n_support_vectors'] for r in brix_results], 'purple',
marker='o')
axes[1,1].set_xlabel('Epsilon')
axes[1,1].set_ylabel('Number of Support Vectors')
axes[1,1].set_title('BRIX - Model Complexity vs Epsilon')
axes[1,1].grid(True, alpha=0.3)

plt.tight_layout()

plt.show()

best_abw_epsilon = max(abw_results, key=lambda x: x['cv_r2_mean'])
best_brix_epsilon = max(brix_results, key=lambda x: x['cv_r2_mean'])

print(f"\n{' '*60}")

print("BEST EPSILON VALUES (based on CV R²):")

```

```

print(f'{'='*60}')

print(f'ABW:      Best    epsilon    =    {best_abw_epsilon['epsilon']}    (CV    R2    =
{best_abw_epsilon['cv_r2_mean']:.4f})")

print(f'BRIX:    Best    epsilon    =    {best_brix_epsilon['epsilon']}    (CV    R2    =
{best_brix_epsilon['cv_r2_mean']:.4f})")

```

Polynomial Regression

```

import pandas as pd

from sklearn.model_selection import KFold, cross_val_score
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression

df = pd.read_csv('points.csv')
X = df[['temperature', 'density', 'refractive index']]
y_abw = df['ABW']
y_brix = df['brix']

poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

scaler = StandardScaler()
X_poly_scaled = scaler.fit_transform(X_poly)

kf = KFold(n_splits=5, shuffle=True, random_state=42)

def print_cv_scores(X_data, y, label):
    model = LinearRegression()

    mse = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_squared_error')

    r2 = cross_val_score(model, X_data, y, cv=kf, scoring='r2')

```

```

mae = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_absolute_error')

print(f'--- {label} (5-Fold CV) ---')

print(f'MSE: {mse.mean():.4f} (std: {mse.std():.4f})')

print(f'R^2: {r2.mean():.4f} (std: {r2.std():.4f})')

print(f'MAE: {mae.mean():.4f} (std: {mae.std():.4f})\n')


print("### UN-SCALED 2 degree POLYNOMIAL FEATURES ###")

print_cv_scores(X_poly, y_abw, 'ABW (Unscaled)')

print_cv_scores(X_poly, y_brix, 'Brix (Unscaled)')


print("### SCALED 2 degree POLYNOMIAL FEATURES ###")

print_cv_scores(X_poly_scaled, y_abw, 'ABW (Scaled)')

print_cv_scores(X_poly_scaled, y_brix, 'Brix (Scaled)')

import pandas as pd

from sklearn.model_selection import KFold, cross_val_score

from sklearn.preprocessing import PolynomialFeatures, StandardScaler

from sklearn.linear_model import LinearRegression


df = pd.read_csv('points.csv')


X = df[['temperature', 'density', 'refractive index']]

y_abw = df['ABW']

y_brix = df['brix']

```

```
poly = PolynomialFeatures(degree=3, include_bias=False)
```

```
X_poly = poly.fit_transform(X)
```

```
scaler = StandardScaler()
```

```
X_poly_scaled = scaler.fit_transform(X_poly)
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
def print_cv_scores(X_data, y, label):
```

```
    model = LinearRegression()
```

```
    mse = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_squared_error')
```

```
    r2 = cross_val_score(model, X_data, y, cv=kf, scoring='r2')
```

```
    mae = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_absolute_error')
```

```
    print(f'--- {label} (5-Fold CV) ---')
```

```
    print(f'MSE: {mse.mean():.4f} (std: {mse.std():.4f})')
```

```
    print(f'R^2: {r2.mean():.4f} (std: {r2.std():.4f})')
```

```
    print(f'MAE: {mae.mean():.4f} (std: {mae.std():.4f})\n')
```

```
print("### UN-SCALED 3 degree POLYNOMIAL FEATURES ###")
```

```
print_cv_scores(X_poly, y_abw, 'ABW (Unscaled)')
```

```
print_cv_scores(X_poly, y_brix, 'Brix (Unscaled)')
```

```
print("### SCALED 3 degree POLYNOMIAL FEATURES ###")
```

```

print_cv_scores(X_poly_scaled, y_abw, 'ABW (Scaled)')
print_cv_scores(X_poly_scaled, y_brix, 'Brix (Scaled)')

import pandas as pd

from sklearn.model_selection import KFold, cross_val_score
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression


df = pd.read_csv('points.csv')


X = df[['temperature', 'density', 'refractive index']]
y_abw = df['ABW']
y_brix = df['brix']


poly = PolynomialFeatures(degree=4, include_bias=False)
X_poly = poly.fit_transform(X)


scaler = StandardScaler()
X_poly_scaled = scaler.fit_transform(X_poly)


kf = KFold(n_splits=5, shuffle=True, random_state=42)


def print_cv_scores(X_data, y, label):
    model = LinearRegression()
    mse = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_squared_error')

```

```

r2 = cross_val_score(model, X_data, y, cv=kf, scoring='r2')

mae = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_absolute_error')

print(f'--- {label} (5-Fold CV) ---')

print(f'MSE: {mse.mean():.4f} (std: {mse.std():.4f})')

print(f'R^2: {r2.mean():.4f} (std: {r2.std():.4f})')

print(f'MAE: {mae.mean():.4f} (std: {mae.std():.4f})\n')


print("### UN-SCALED 4 degree POLYNOMIAL FEATURES ###")

print_cv_scores(X_poly, y_abw, 'ABW (Unscaled)')

print_cv_scores(X_poly, y_brix, 'Brix (Unscaled)')


print("### SCALED 4 degree POLYNOMIAL FEATURES ###")

print_cv_scores(X_poly_scaled, y_abw, 'ABW (Scaled)')

print_cv_scores(X_poly_scaled, y_brix, 'Brix (Scaled)')

import pandas as pd

from sklearn.model_selection import KFold, cross_val_score

from sklearn.preprocessing import PolynomialFeatures, StandardScaler

from sklearn.linear_model import LinearRegression


df = pd.read_csv('some points gone.csv')


X = df[['temperature', 'density', 'refractive index']]

y_abw = df['ABW']

y_brix = df['brix']


poly = PolynomialFeatures(degree=5, include_bias=False)

```

```
X_poly = poly.fit_transform(X)
```

```
scaler = StandardScaler()
```

```
X_poly_scaled = scaler.fit_transform(X_poly)
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
def print_cv_scores(X_data, y, label):
```

```
    model = LinearRegression()
```

```
    mse = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_squared_error')
```

```
    r2 = cross_val_score(model, X_data, y, cv=kf, scoring='r2')
```

```
    mae = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_absolute_error')
```

```
    print(f'--- {label} (5-Fold CV) ---')
```

```
    print(f'MSE: {mse.mean():.4f} (std: {mse.std():.4f})')
```

```
    print(f'R^2: {r2.mean():.4f} (std: {r2.std():.4f})')
```

```
    print(f'MAE: {mae.mean():.4f} (std: {mae.std():.4f})\n')
```

```
print("### UN-SCALED 5 degree POLYNOMIAL FEATURES ###")
```

```
print_cv_scores(X_poly, y_abw, 'ABW (Unscaled)')
```

```
print_cv_scores(X_poly, y_brix, 'Brix (Unscaled)')
```

```
print("### SCALED 5 degree POLYNOMIAL FEATURES ###")
```

```
print_cv_scores(X_poly_scaled, y_abw, 'ABW (Scaled)')
```

```
print_cv_scores(X_poly_scaled, y_brix, 'Brix (Scaled)')
```

```
import pandas as pd
```

```
from sklearn.model_selection import KFold, cross_val_score
```

```

from sklearn.preprocessing import PolynomialFeatures, StandardScaler

from sklearn.linear_model import LinearRegression


df = pd.read_csv('points.csv')


X = df[['temperature', 'density', 'refractive index']]
y_abw = df['ABW']
y_brix = df['brix']


poly = PolynomialFeatures(degree=6, include_bias=False)
X_poly = poly.fit_transform(X)


scaler = StandardScaler()
X_poly_scaled = scaler.fit_transform(X_poly)


kf = KFold(n_splits=5, shuffle=True, random_state=42)


def print_cv_scores(X_data, y, label):
    model = LinearRegression()

    mse = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_squared_error')
    r2 = cross_val_score(model, X_data, y, cv=kf, scoring='r2')
    mae = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_absolute_error')

    print(f'--- {label} (5-Fold CV) ---')

    print(f'MSE: {mse.mean():.4f} (std: {mse.std():.4f})')

    print(f'R^2: {r2.mean():.4f} (std: {r2.std():.4f})')

    print(f'MAE: {mae.mean():.4f} (std: {mae.std():.4f})\n')

```

```

print("### UN-SCALED 6 degree POLYNOMIAL FEATURES ###")
print_cv_scores(X_poly, y_abw, 'ABW (Unscaled)')
print_cv_scores(X_poly, y_brix, 'Brix (Unscaled)')

print("### SCALED 6 degreePOLYNOMIAL FEATURES ###")
print_cv_scores(X_poly_scaled, y_abw, 'ABW (Scaled)')
print_cv_scores(X_poly_scaled, y_brix, 'Brix (Scaled)')

import pandas as pd

from sklearn.model_selection import KFold, cross_val_score
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt

import numpy as np

df = pd.read_csv('points.csv')

X = df[['temperature', 'density', 'refractive index']]
y_abw = df['ABW']
y_brix = df['brix']

kf = KFold(n_splits=5, shuffle=True, random_state=42)

def get_cv_scores(X_data, y):

```

```

"""Get cross-validation scores and return mean values"""

model = LinearRegression()

mse = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_squared_error')

r2 = cross_val_score(model, X_data, y, cv=kf, scoring='r2')

mae = -cross_val_score(model, X_data, y, cv=kf, scoring='neg_mean_absolute_error')

return mse.mean(), r2.mean(), mae.mean()


degrees = range(1, 16)

results = {

    'abw_unscaled': {'mse': [], 'r2': [], 'mae': []},

    'brix_unscaled': {'mse': [], 'r2': [], 'mae': []},

    'abw_scaled': {'mse': [], 'r2': [], 'mae': []},

    'brix_scaled': {'mse': [], 'r2': [], 'mae': []}

}


print("Testing polynomial degrees 1 to 25...")

for degree in degrees:

    print(f"Processing degree {degree}...")


    poly = PolynomialFeatures(degree=degree, include_bias=False)

    X_poly = poly.fit_transform(X)


    scaler = StandardScaler()

    X_poly_scaled = scaler.fit_transform(X_poly)

```

```

mse_abw, r2_abw, mae_abw = get_cv_scores(X_poly, y_abw)
mse_brix, r2_brix, mae_brix = get_cv_scores(X_poly, y_brix)

results['abw_unscaled']['mse'].append(mse_abw)
results['abw_unscaled']['r2'].append(r2_abw)
results['abw_unscaled']['mae'].append(mae_abw)

results['brix_unscaled']['mse'].append(mse_brix)
results['brix_unscaled']['r2'].append(r2_brix)
results['brix_unscaled']['mae'].append(mae_brix)

mse_abw_s, r2_abw_s, mae_abw_s = get_cv_scores(X_poly_scaled, y_abw)
mse_brix_s, r2_brix_s, mae_brix_s = get_cv_scores(X_poly_scaled, y_brix)

results['abw_scaled']['mse'].append(mse_abw_s)
results['abw_scaled']['r2'].append(r2_abw_s)
results['abw_scaled']['mae'].append(mae_abw_s)

results['brix_scaled']['mse'].append(mse_brix_s)
results['brix_scaled']['r2'].append(r2_brix_s)
results['brix_scaled']['mae'].append(mae_brix_s)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('Polynomial Regression Performance vs Degree', fontsize=16)

metrics = ['mse', 'r2', 'mae']

```

```
metric_names = ['Mean Squared Error', 'R2 Score', 'Mean Absolute Error']
```

```
for i, (metric, name) in enumerate(zip(metrics, metric_names)):
```

```
    # ABW plots
```

```
    axes[0, i].plot(degrees, results['abw_unscaled'][metric], 'b-o', label='Unscaled', markersize=4)
```

```
    axes[0, i].plot(degrees, results['abw_scaled'][metric], 'r-s', label='Scaled', markersize=4)
```

```
    axes[0, i].set_title(f'ABW - {name}')
```

```
    axes[0, i].set_xlabel('Polynomial Degree')
```

```
    axes[0, i].set_ylabel(name)
```

```
    axes[0, i].legend()
```

```
    axes[0, i].grid(True, alpha=0.3)
```

```
    # Brix plots
```

```
    axes[1, i].plot(degrees, results['brix_unscaled'][metric], 'b-o', label='Unscaled', markersize=4)
```

```
    axes[1, i].plot(degrees, results['brix_scaled'][metric], 'r-s', label='Scaled', markersize=4)
```

```
    axes[1, i].set_title(f'Brix Polynomial Regression:{name}')
```

```
    axes[1, i].set_xlabel('Polynomial Degree')
```

```
    axes[1, i].set_ylabel(name)
```

```
    axes[1, i].legend()
```

```
    axes[1, i].grid(True, alpha=0.3)
```

```
plt.tight_layout()
```

```
plt.show()
```

```
print("\n" + "="*50)
```

```
print("OPTIMAL DEGREES SUMMARY")
```

```

print("="*50)

for target in ['abw', 'brix']:
    for scaling in ['unscaled', 'scaled']:
        key = f"{target}_{scaling}"

        # Best R² (highest)
        best_r2_idx = np.argmax(results[key]['r2'])
        best_r2_degree = degrees[best_r2_idx]
        best_r2_value = results[key]['r2'][best_r2_idx]

        # Best MAE (lowest)
        best_mae_idx = np.argmin(results[key]['mae'])
        best_mae_degree = degrees[best_mae_idx]
        best_mae_value = results[key]['mae'][best_mae_idx]

        # Best MSE (lowest)
        best_mse_idx = np.argmin(results[key]['mse'])
        best_mse_degree = degrees[best_mse_idx]
        best_mse_value = results[key]['mse'][best_mse_idx]

    print(f"\n{target.upper()} ({scaling.upper()}):")
    print(f" Best R² Score: Degree {best_r2_degree} (R² = {best_r2_value:.4f})")
    print(f" Best MAE: Degree {best_mae_degree} (MAE = {best_mae_value:.4f})")
    print(f" Best MSE: Degree {best_mse_degree} (MSE = {best_mse_value:.4f})")

```

```

print("\n" + "="*50)

print("DETAILED RESULTS TABLE")

print("="*50)


summary_data = []

for i, degree in enumerate(degrees):

    summary_data.append({

        'Degree': degree,

        'ABW_Unscaled_R2': results['abw_unscaled']['r2'][i],

        'ABW_Unscaled_MAE': results['abw_unscaled']['mae'][i],

        'ABW_Scaled_R2': results['abw_scaled']['r2'][i],

        'ABW_Scaled_MAE': results['abw_scaled']['mae'][i],

        'Brix_Unscaled_R2': results['brix_unscaled']['r2'][i],

        'Brix_Unscaled_MAE': results['brix_unscaled']['mae'][i],

        'Brix_Scaled_R2': results['brix_scaled']['r2'][i],

        'Brix_Scaled_MAE': results['brix_scaled']['mae'][i],

    })


summary_df = pd.DataFrame(summary_data)

print(summary_df.round(4))

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import PolynomialFeatures, StandardScaler

from sklearn.linear_model import LinearRegression

```

```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

```
df = pd.read_csv('oints.csv')
```

```
X = df[['temperature', 'density', 'refractive index']]
```

```
y_abw = df['ABW']
```

```
y_brix = df['brix']
```

```
X_train, X_test, y_abw_train, y_abw_test = train_test_split(  
    X, y_abw, test_size=0.2, random_state=42  
)
```

```
_, _, y_brix_train, y_brix_test = train_test_split(  
    X, y_brix, test_size=0.2, random_state=42  
)
```

```
poly = PolynomialFeatures(degree=8, include_bias=False)
```

```
X_train_poly = poly.fit_transform(X_train)
```

```
X_test_poly = poly.transform(X_test)
```

```
scaler = StandardScaler()
```

```
X_train_poly_scaled = scaler.fit_transform(X_train_poly)
```

```
X_test_poly_scaled = scaler.transform(X_test_poly)
```

```
model_abw = LinearRegression()
```

```
model_brix = LinearRegression()
```

```

model_abw.fit(X_train_poly_scaled, y_abw_train)
model_brix.fit(X_train_poly_scaled, y_brix_train)

y_abw_train_pred = model_abw.predict(X_train_poly_scaled)
y_abw_test_pred = model_abw.predict(X_test_poly_scaled)

y_brix_train_pred = model_brix.predict(X_train_poly_scaled)
y_brix_test_pred = model_brix.predict(X_test_poly_scaled)

def print_metrics(y_true, y_pred, dataset_name, target_name):
    r2 = r2_score(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    print(f"{target_name} - {dataset_name}:")
    print(f" R2 = {r2:.4f}")
    print(f" MSE = {mse:.4f}")
    print(f" MAE = {mae:.4f}")
    print()

print("=== MODEL PERFORMANCE METRICS ===")
print_metrics(y_abw_train, y_abw_train_pred, "Training", "ABW")
print_metrics(y_abw_test, y_abw_test_pred, "Testing", "ABW")
print_metrics(y_brix_train, y_brix_train_pred, "Training", "Brix")
print_metrics(y_brix_test, y_brix_test_pred, "Testing", "Brix")

```

```

fig, axes = plt.subplots(2, 2, figsize=(15, 12))

fig.suptitle('Predicted vs Actual Values - Polynomial Regression (Degree 8)', fontsize=16)

# Plot 1: ABW Training
axes[0, 0].scatter(y_abw_train, y_abw_train_pred, alpha=0.6, color='blue')
axes[0, 0].plot([y_abw_train.min(), y_abw_train.max()],
                [y_abw_train.min(), y_abw_train.max()], 'r--', lw=2)
axes[0, 0].set_xlabel('Actual ABW')
axes[0, 0].set_ylabel('Predicted ABW')
axes[0, 0].set_title(f'ABW - Training Set\nR2 = {r2_score(y_abw_train, y_abw_train_pred):.4f}')
axes[0, 0].grid(True, alpha=0.3)

# Plot 2: ABW Testing
axes[0, 1].scatter(y_abw_test, y_abw_test_pred, alpha=0.6, color='green')
axes[0, 1].plot([y_abw_test.min(), y_abw_test.max()],
                [y_abw_test.min(), y_abw_test.max()], 'r--', lw=2)
axes[0, 1].set_xlabel('Actual ABW')
axes[0, 1].set_ylabel('Predicted ABW')
axes[0, 1].set_title(f'ABW - Testing Set\nR2 = {r2_score(y_abw_test, y_abw_test_pred):.4f}')
axes[0, 1].grid(True, alpha=0.3)

# Plot 3: Brix Training
axes[1, 0].scatter(y_brix_train, y_brix_train_pred, alpha=0.6, color='purple')
axes[1, 0].plot([y_brix_train.min(), y_brix_train.max()],
                [y_brix_train.min(), y_brix_train.max()], 'r--', lw=2)
axes[1, 0].set_xlabel('Actual Brix')

```

```

axes[1, 0].set_ylabel('Predicted Brix')

axes[1, 0].set_title(f'Brix - Training Set - Predicted vs Actual Values\nR2 = {r2_score(y_brix_train,
y_brix_train_pred):.4f}')

axes[1, 0].grid(True, alpha=0.3)


# Plot 4: Brix Testing

axes[1, 1].scatter(y_brix_test, y_brix_test_pred, alpha=0.6, color='blue')

axes[1, 1].plot([y_brix_test.min(), y_brix_test.max()],
                [y_brix_test.min(), y_brix_test.max()], 'r--', lw=2)

axes[1, 1].set_xlabel('Actual Brix')

axes[1, 1].set_ylabel('Predicted Brix')

axes[1, 1].set_title(f'Brix - Testing Set - Predicted vs Actual Values\nR2 = {r2_score(y_brix_test,
y_brix_test_pred):.4f}')

axes[1, 1].grid(True, alpha=0.3)


plt.tight_layout()

plt.show()


fig, axes = plt.subplots(2, 2, figsize=(15, 12))

fig.suptitle('Residual Plots - Predicted vs Residuals', fontsize=16)


# ABW Training residuals

abw_train_residuals = y_abw_train - y_abw_train_pred

axes[0, 0].scatter(y_abw_train_pred, abw_train_residuals, alpha=0.6, color='blue')

axes[0, 0].axhline(y=0, color='r', linestyle='--')

```

```

axes[0, 0].set_xlabel('Predicted ABW')
axes[0, 0].set_ylabel('Residuals')
axes[0, 0].set_title('ABW - Training Residuals')
axes[0, 0].grid(True, alpha=0.3)

# ABW Testing residuals
abw_test_residuals = y_abw_test - y_abw_test_pred
axes[0, 1].scatter(y_abw_test_pred, abw_test_residuals, alpha=0.6, color='green')
axes[0, 1].axhline(y=0, color='r', linestyle='--')
axes[0, 1].set_xlabel('Predicted ABW')
axes[0, 1].set_ylabel('Residuals')
axes[0, 1].set_title('ABW - Testing Residuals')
axes[0, 1].grid(True, alpha=0.3)

# Brix Training residuals
brix_train_residuals = y_brix_train - y_brix_train_pred
axes[1, 0].scatter(y_brix_train_pred, brix_train_residuals, alpha=0.6, color='purple')
axes[1, 0].axhline(y=0, color='r', linestyle='--')
axes[1, 0].set_xlabel('Predicted Brix')
axes[1, 0].set_ylabel('Residuals')
axes[1, 0].set_title('Brix - Training Residuals')
axes[1, 0].grid(True, alpha=0.3)

# Brix Testing residuals
brix_test_residuals = y_brix_test - y_brix_test_pred
axes[1, 1].scatter(y_brix_test_pred, brix_test_residuals, alpha=0.6, color='orange')

```

```

axes[1, 1].axhline(y=0, color='r', linestyle='--')
axes[1, 1].set_xlabel('Predicted Brix')
axes[1, 1].set_ylabel('Residuals')
axes[1, 1].set_title('Brix - Testing Residuals')
axes[1, 1].grid(True, alpha=0.3)

```

```

plt.tight_layout()
plt.show()

```

```

def analyze_feature_importance(model, poly_features, feature_names, model_name):

```

```

    """

```

```

    Analyze feature importance for polynomial regression model

```

```

    """

```

```

    print(f"\n=== FEATURE IMPORTANCE ANALYSIS - {model_name} ===")

```

```

    coefficients = model.coef_

```

```

    # Get feature names from polynomial features

```

```

    poly_feature_names = poly_features.get_feature_names_out(feature_names)

```

```

    coef_df = pd.DataFrame({

```

```

        'feature': poly_feature_names,

```

```

        'coefficient': coefficients,

```

```

        'abs_coefficient': np.abs(coefficients)
    })

    coef_df = coef_df.sort_values('abs_coefficient', ascending=False)

    print(f"Total number of polynomial features: {len(poly_feature_names)}")
    print(f"\nTop 20 most important features (by absolute coefficient):")
    print(coef_df.head(20))

    return coef_df

feature_names = ['temperature', 'density', 'refractive index']

coef_df_abw = analyze_feature_importance(model_abw, poly, feature_names, "ABW")
coef_df_brix = analyze_feature_importance(model_brix, poly, feature_names, "Brix")

def plot_feature_importance(coef_df, model_name, top_n=20):
    """
    Plot feature importance
    """
    top_features = coef_df.head(top_n)

    plt.figure(figsize=(12, 8))

    colors = ['red' if coef < 0 else 'blue' for coef in top_features['coefficient']]

    plt.barh(range(len(top_features)), top_features['coefficient'], color=colors, alpha=0.7)

```

```
plt.yticks(range(len(top_features)), top_features['feature'], fontsize=8)
plt.xlabel('Coefficient Value')
plt.title(f'Top {top_n} Feature Coefficients - {model_name}')
plt.grid(True, alpha=0.3, axis='x')
```

```
plt.axvline(x=0, color='black', linestyle='-', alpha=0.5)
from matplotlib.patches import Patch
legend_elements = [Patch(facecolor='blue', alpha=0.7, label='Positive'),
                    Patch(facecolor='red', alpha=0.7, label='Negative')]
plt.legend(handles=legend_elements)
```

```
plt.tight_layout()
plt.show()
```

```
plot_feature_importance(coef_df_abw, "ABW Model")
plot_feature_importance(coef_df_brix, "Brix Model")
def predict_solution_clean(temperature, density, refractive_index):
    """Clean prediction function without warnings"""
```

```
input_df = pd.DataFrame({
    'temperature': [temperature],
    'density': [density],
    'refractive index': [refractive_index]
})
```

```

input_poly = poly.transform(input_df)
input_scaled = scaler.transform(input_poly)

predicted_abw = model_abw.predict(input_scaled)[0]
predicted_brix = model_brix.predict(input_scaled)[0]

return {
    'ABW': round(predicted_abw, 2),
    'Brix': round(predicted_brix, 2)
}

result = predict_solution_clean(24.0, 1.03419, 1.3683)
print(f"ABW: {result['ABW']}%, Brix: {result['Brix']}%")

result = predict_solution_clean(20.0, 0.94823, 1.35528)
print(f"ABW: {result['ABW']}%, Brix: {result['Brix']}%")

```

Neural Networks

```

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

from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.multioutput import MultiOutputRegressor
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('points.csv')
```

```
X = df[['temperature', 'density', 'refractive index']]
y = df[['ABW', 'brix']]
```

```
print("Dataset shape:", df.shape)
print("\nFirst few rows:")
print(df.head())
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

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

```
print(f"\nTraining set size: {X_train.shape[0]}")
print(f"\nTest set size: {X_test.shape[0]}")
```

```
results = {
```

```

'scaled': {},
'unscaled': {}
}

```

```

nn_configs = [
    {'name': 'Small NN (10)', 'hidden_layer_sizes': (10,)},
    {'name': 'Medium NN (50, 25)', 'hidden_layer_sizes': (50, 25)},
    {'name': 'Large NN (100, 50)', 'hidden_layer_sizes': (100, 50)},
    {'name': 'Deep NN (100, 50, 25)', 'hidden_layer_sizes': (100, 50, 25)},
    {'name': 'Your Config (50, 100)', 'hidden_layer_sizes': (50, 100)},
    {'name': 'Your Config (2, 4)', 'hidden_layer_sizes': (2, 4)}
]

```

```

def evaluate_model(model, X_test, y_test, y_pred):
    """Calculate evaluation metrics"""

    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    mse_abw = mean_squared_error(y_test.iloc[:, 0], y_pred[:, 0])
    mse_brix = mean_squared_error(y_test.iloc[:, 1], y_pred[:, 1])
    r2_abw = r2_score(y_test.iloc[:, 0], y_pred[:, 0])
    r2_brix = r2_score(y_test.iloc[:, 1], y_pred[:, 1])

    return {

```

```

'mse_overall': mse,

'mae_overall': mae,

'r2_overall': r2,

'mse_abw': mse_abw,

'mse_brix': mse_brix,

'r2_abw': r2_abw,

'r2_brix': r2_brix
}

```

```
def plot_predictions(y_true, y_pred, title, data_type):
```

```
    """Plot predicted vs actual values"""
```

```
    fig, axes = plt.subplots(1, 2, figsize=(15, 6))
```

```
    # ABW
```

```
    axes[0].scatter(y_true.iloc[:, 0], y_pred[:, 0], alpha=0.6, color='blue')
```

```
    axes[0].plot([y_true.iloc[:, 0].min(), y_true.iloc[:, 0].max()],
```

```
                  [y_true.iloc[:, 0].min(), y_true.iloc[:, 0].max()], 'r--', lw=2)
```

```
    axes[0].set_xlabel('Actual ABW')
```

```
    axes[0].set_ylabel('Predicted ABW')
```

```
    axes[0].set_title(f'{title} - ABW ({data_type})')
```

```
    axes[0].grid(True, alpha=0.3)
```

```
    # Brix
```

```
    axes[1].scatter(y_true.iloc[:, 1], y_pred[:, 1], alpha=0.6, color='green')
```

```
    axes[1].plot([y_true.iloc[:, 1].min(), y_true.iloc[:, 1].max()],
```

```
                  [y_true.iloc[:, 1].min(), y_true.iloc[:, 1].max()], 'r--', lw=2)
```

```
axes[1].set_xlabel('Actual Brix')
axes[1].set_ylabel('Predicted Brix')
axes[1].set_title(f'{title} - Brix ({data_type})')
axes[1].grid(True, alpha=0.3)
```

```
plt.tight_layout()
plt.show()
```

```
print("SCALED DATA EXPERIMENTS")
```

```
for config in nn_configs:
```

```
    print(f"\nTesting {config['name']}...")
```

```
    model = MLPRegressor(
        hidden_layer_sizes=config['hidden_layer_sizes'],
        max_iter=10000,
        random_state=42,
        early_stopping=True,
        validation_fraction=0.1
    )
```

```
    model.fit(X_train_scaled, y_train)
```

```
y_pred_train = model.predict(X_train_scaled)
```

```
y_pred_test = model.predict(X_test_scaled)
```

```
train_metrics = evaluate_model(model, X_train, y_train, y_pred_train)
```

```
test_metrics = evaluate_model(model, X_test, y_test, y_pred_test)
```

```
results['scaled'][config['name']] = {
```

```
    'model': model,
```

```
    'train_metrics': train_metrics,
```

```
    'test_metrics': test_metrics,
```

```
    'y_pred_train': y_pred_train,
```

```
    'y_pred_test': y_pred_test
```

```
}
```

```
print(f"Training MSE: {train_metrics['mse_overall']:.6f}, R²: {train_metrics['r2_overall']:.4f}")
```

```
print(f"Test MSE: {test_metrics['mse_overall']:.6f}, R²: {test_metrics['r2_overall']:.4f}")
```

```
print(f"Test MAE: {test_metrics['mae_overall']:.6f}")
```

```
plot_predictions(y_train, y_pred_train, config['name'], "Training - Scaled")
```

```
plot_predictions(y_test, y_pred_test, config['name'], "Test - Scaled")
```

p

```
print("UNSCALED DATA EXPERIMENTS")
```

```
for config in nn_configs:
```

```
    model = MLPRegressor(  
        hidden_layer_sizes=config['hidden_layer_sizes'],  
        max_iter=10000,  
        random_state=42,  
        early_stopping=True,  
        validation_fraction=0.1  
    )
```

```
    model.fit(X_train, y_train)
```

```
    y_pred_train = model.predict(X_train)
```

```
    y_pred_test = model.predict(X_test)
```

```
    train_metrics = evaluate_model(model, X_train, y_train, y_pred_train)
```

```
    test_metrics = evaluate_model(model, X_test, y_test, y_pred_test)
```

```
    results['unscaled'][config['name']] = {  
        'model': model,  
        'train_metrics': train_metrics,  
        'test_metrics': test_metrics,  
        'y_pred_train': y_pred_train,  
        'y_pred_test': y_pred_test  
    }
```

```

print(f"Training MSE: {train_metrics['mse_overall']:.6f}, R²: {train_metrics['r2_overall']:.4f}")
print(f"Test MSE: {test_metrics['mse_overall']:.6f}, R²: {test_metrics['r2_overall']:.4f}")
print(f"Test MAE: {test_metrics['mae_overall']:.6f}")
plot_predictions(y_train, y_pred_train, config['name'], "Training - Unscaled")
plot_predictions(y_test, y_pred_test, config['name'], "Test - Unscaled")

print("K-FOLD CROSS VALIDATION")

kfold = KFold(n_splits=5, shuffle=True, random_state=42)

best_config = nn_configs[2]
print(f"\nTesting {best_config['name']} with K-Fold CV:")
model_scaled = MLPRegressor(
    hidden_layer_sizes=best_config['hidden_layer_sizes'],
    max_iter=10000,
    random_state=42
)

cv_scores_scaled = cross_val_score(model_scaled, X_train_scaled, y_train,
                                     cv=kfold, scoring='neg_mean_squared_error')
cv_r2_scaled = cross_val_score(model_scaled, X_train_scaled, y_train,
                                cv=kfold, scoring='r2')

```

```
print(f"Scaled Data CV MSE: {-cv_scores_scaled.mean():.6f} (±{cv_scores_scaled.std():.6f})")
```

```
print(f"Scaled Data CV R²: {cv_r2_scaled.mean():.4f} (±{cv_r2_scaled.std():.4f})")
```

```
cv_scores_unscaled = cross_val_score(model_scaled, X_train, y_train,  
                                     cv=kfold, scoring='neg_mean_squared_error')
```

```
cv_r2_unscaled = cross_val_score(model_scaled, X_train, y_train,  
                                 cv=kfold, scoring='r2')
```

```
print(f"Unscaled      Data      CV      MSE:      {-cv_scores_unscaled.mean():.6f}  
(±{cv_scores_unscaled.std():.6f})")
```

```
print(f"Unscaled Data CV R²: {cv_r2_unscaled.mean():.4f} (±{cv_r2_unscaled.std():.4f})")
```

```
print("\n" + "="*50)
```

```
print("SUMMARY OF RESULTS")
```

```
print("="*50)
```

```
# Summary comparison
```

```
print("\nBest performing models:")
```

```
print("-" * 40)
```

```
best_scaled = min(results['scaled'].items(),
```

```
                  key=lambda x: x[1]['test_metrics']['mse_overall'])
```

```
print(f"Best Scaled Model: {best_scaled[0]}")
```

```
print(f" Test MSE: {best_scaled[1]['test_metrics']['mse_overall']:.6f}")
```

```
print(f" Test R2: {best_scaled[1]['test_metrics']['r2_overall']:.4f}")
print(f" Test MAE: {best_scaled[1]['test_metrics']['mae_overall']:.6f}")
```

```
best_unscaled = min(results['unscaled'].items(),
                    key=lambda x: x[1]['test_metrics']['mse_overall'])
print(f"\nBest Unscaled Model: {best_unscaled[0]}")
print(f" Test MSE: {best_unscaled[1]['test_metrics']['mse_overall']:.6f}")
print(f" Test R2: {best_unscaled[1]['test_metrics']['r2_overall']:.4f}")
print(f" Test MAE: {best_unscaled[1]['test_metrics']['mae_overall']:.6f}")
```

```
print(f"\nDetailed metrics for best scaled model ({best_scaled[0]}):")
print(f"      ABW      -   MSE:   {best_scaled[1]['test_metrics']['mse_abw']:.6f},   R2:
{best_scaled[1]['test_metrics']['r2_abw']:.4f}")
print(f"      Brix      -   MSE:   {best_scaled[1]['test_metrics']['mse_brix']:.6f},   R2:
{best_scaled[1]['test_metrics']['r2_brix']:.4f}")
```

```
print(f"\nDetailed metrics for best unscaled model ({best_unscaled[0]}):")
print(f"      ABW      -   MSE:   {best_unscaled[1]['test_metrics']['mse_abw']:.6f},   R2:
{best_unscaled[1]['test_metrics']['r2_abw']:.4f}")
print(f"      Brix      -   MSE:   {best_unscaled[1]['test_metrics']['mse_brix']:.6f},   R2:
{best_unscaled[1]['test_metrics']['r2_brix']:.4f}")
```

K Nearest Neighbors

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score, KFold

from sklearn.neighbors import KNeighborsRegressor

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

from sklearn.multioutput import MultiOutputRegressor

import warnings


np.random.seed(42)


plt.style.use('default')

sns.set_palette("husl")

df = pd.read_csv('points.csv')

np.random.seed(42)

print(f"Dataset shape: {df.shape}")

print("\nDataset info:")

print(df.describe())

X = df[['temperature', 'density', 'refractive_index']]

y = df[['ABW', 'brix']]
```

```

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"Training set size: {X_train.shape[0]}")
print(f"Testing set size: {X_test.shape[0]}")

scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

y_train_scaled = scaler_y.fit_transform(y_train)
y_test_scaled = scaler_y.transform(y_test)

def evaluate_model(y_true, y_pred, model_name, data_type):
    """Evaluate model performance and return metrics"""
    mse_abw = mean_squared_error(y_true.iloc[:, 0], y_pred[:, 0])
    mse_brix = mean_squared_error(y_true.iloc[:, 1], y_pred[:, 1])

    r2_abw = r2_score(y_true.iloc[:, 0], y_pred[:, 0])
    r2_brix = r2_score(y_true.iloc[:, 1], y_pred[:, 1])

    mae_abw = mean_absolute_error(y_true.iloc[:, 0], y_pred[:, 0])
    mae_brix = mean_absolute_error(y_true.iloc[:, 1], y_pred[:, 1])

```

```

print(f"\n{model_name} - {data_type} Performance:")

print(f"ABW - MSE: {mse_abw:.4f}, R2: {r2_abw:.4f}, MAE: {mae_abw:.4f}")

print(f"Brix - MSE: {mse_brix:.4f}, R2: {r2_brix:.4f}, MAE: {mae_brix:.4f}")


return {

    'mse_abw': mse_abw, 'mse_brix': mse_brix,

    'r2_abw': r2_abw, 'r2_brix': r2_brix,

    'mae_abw': mae_abw, 'mae_brix': mae_brix

}


def plot_predictions(y_true, y_pred, title, data_type):

    """Create predicted vs actual plots"""

    fig, axes = plt.subplots(1, 2, figsize=(15, 6))

    # ABW

    axes[0].scatter(y_true.iloc[:, 0], y_pred[:, 0], alpha=0.6, s=30)

    axes[0].plot([y_true.iloc[:, 0].min(), y_true.iloc[:, 0].max()],

                 [y_true.iloc[:, 0].min(), y_true.iloc[:, 0].max()], 'r--', lw=2)

    axes[0].set_xlabel('Actual ABW')

    axes[0].set_ylabel('Predicted ABW')

    axes[0].set_title(f'{title} - ABW ({data_type})')

    axes[0].grid(True, alpha=0.3)

    # Brix plot

    axes[1].scatter(y_true.iloc[:, 1], y_pred[:, 1], alpha=0.6, s=30)

```

```

axes[1].plot([y_true.iloc[:, 1].min(), y_true.iloc[:, 1].max()],
             [y_true.iloc[:, 1].min(), y_true.iloc[:, 1].max()], 'r--', lw=2)
axes[1].set_xlabel('Actual Brix')
axes[1].set_ylabel('Predicted Brix')
axes[1].set_title(f'{title} - Brix ({data_type})')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()

plt.show()

k_values = range(1, 21)
k_scores = []

print("\nFinding optimal k value...")
for k in k_values:
    knn = MultiOutputRegressor(KNeighborsRegressor(n_neighbors=k))
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)

    r2_abw = r2_score(y_test.iloc[:, 0], y_pred[:, 0])
    r2_brix = r2_score(y_test.iloc[:, 1], y_pred[:, 1])
    avg_r2 = (r2_abw + r2_brix) / 2
    k_scores.append(avg_r2)

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

plt.plot(k_values, k_scores, 'bo-', linewidth=2, markersize=8)

```

```

plt.xlabel('k (Number of Neighbors)')
plt.ylabel('Average R2 Score')
plt.title('KNN Performance vs k Value')
plt.grid(True, alpha=0.3)
plt.show()

optimal_k = k_values[np.argmax(k_scores)]
print(f"Optimal k value: {optimal_k}")

# Initialize models with optimal k
knn_unscaled = MultiOutputRegressor(KNeighborsRegressor(n_neighbors=optimal_k))
knn_scaled = MultiOutputRegressor(KNeighborsRegressor(n_neighbors=optimal_k))

print(f"\n{'='*60}")
print("KNN REGRESSION ANALYSIS")
print(f"{'='*60}")

# UNSCALED DATA
print(f"\n{'='*40}")
print("1. UNSCALED DATA ANALYSIS")
print(f"{'='*40}")

knn_unscaled.fit(X_train, y_train)

y_train_pred_unscaled = knn_unscaled.predict(X_train)
y_test_pred_unscaled = knn_unscaled.predict(X_test)

```

```

train_metrics_unscaled = evaluate_model(y_train, y_train_pred_unscaled, "KNN Unscaled",
"Training")

test_metrics_unscaled = evaluate_model(y_test, y_test_pred_unscaled, "KNN Unscaled",
"Testing")

plot_predictions(y_train, y_train_pred_unscaled, "KNN Unscaled", "Training Data")

plot_predictions(y_test, y_test_pred_unscaled, "KNN Unscaled", "Testing Data")

```

```

# SCALED DATA

```

```

print(f"\n{' '*40}")

print("2. SCALED DATA ANALYSIS")

print(f"{' '*40}")

knn_scaled.fit(X_train_scaled, y_train)

y_train_pred_scaled = knn_scaled.predict(X_train_scaled)

y_test_pred_scaled = knn_scaled.predict(X_test_scaled)

```

```

train_metrics_scaled = evaluate_model(y_train, y_train_pred_scaled, "KNN Scaled", "Training")

test_metrics_scaled = evaluate_model(y_test, y_test_pred_scaled, "KNN Scaled", "Testing")

plot_predictions(y_train, y_train_pred_scaled, "KNN Scaled", "Training Data")

plot_predictions(y_test, y_test_pred_scaled, "KNN Scaled", "Testing Data")

```

```

print(f"\n{' '*40}")

print("3. K-FOLD CROSS-VALIDATION")

print(f"{' '*40}")

kfold = KFold(n_splits=5, shuffle=True, random_state=42)

print("\nUnscaled Data Cross-Validation:")

```

```

cv_scores_unscaled = []

for target_idx, target_name in enumerate(['ABW', 'Brix']):

    knn_single = KNeighborsRegressor(n_neighbors=optimal_k)

    cv_scores = cross_val_score(knn_single, X, y.iloc[:, target_idx],
                                cv=kfold, scoring='r2')

    cv_scores_unscaled.append(cv_scores)

    print(f'{target_name} - Mean CV R2: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})')


print("\nScaled Data Cross-Validation:")

X_scaled_full = scaler_X.fit_transform(X)

cv_scores_scaled = []

for target_idx, target_name in enumerate(['ABW', 'Brix']):

    knn_single = KNeighborsRegressor(n_neighbors=optimal_k)

    cv_scores = cross_val_score(knn_single, X_scaled_full, y.iloc[:, target_idx],
                                cv=kfold, scoring='r2')

    cv_scores_scaled.append(cv_scores)

    print(f'{target_name} - Mean CV R2: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})')


print(f'\n{'='*40}')

print("4. PERFORMANCE COMPARISON SUMMARY")

print(f'\n{'='*40}')

```

```

comparison_data = {

    'Model': ['Unscaled', 'Scaled'],

    'Test_R2_ABW': [test_metrics_unscaled['r2_abw'], test_metrics_scaled['r2_abw']],

    'Test_R2_Brix': [test_metrics_unscaled['r2_brix'], test_metrics_scaled['r2_brix']],

```

```

'Test_MSE_ABW': [test_metrics_unscaled['mse_abw'], test_metrics_scaled['mse_abw']],
'Test_MSE_Brix': [test_metrics_unscaled['mse_brix'], test_metrics_scaled['mse_brix']],
'Test_MAE_ABW': [test_metrics_unscaled['mae_abw'], test_metrics_scaled['mae_abw']],
'Test_MAE_Brix': [test_metrics_unscaled['mae_brix'], test_metrics_scaled['mae_brix']],
'CV_R2_ABW': [cv_scores_unscaled[0].mean(), cv_scores_scaled[0].mean()],
'CV_R2_Brix': [cv_scores_unscaled[1].mean(), cv_scores_scaled[1].mean()]
}

```

```

comparison_df = pd.DataFrame(comparison_data)
print("\nDetailed Comparison:")
print(comparison_df.round(4))
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
metrics = ['Test_R2_ABW', 'Test_R2_Brix', 'CV_R2_ABW', 'CV_R2_Brix']
titles = ['Test R2 - ABW', 'Test R2 - Brix', 'CV R2 - ABW', 'CV R2 - Brix']

```

```

for i, (metric, title) in enumerate(zip(metrics, titles)):
    ax = axes[i//2, i%2]
    ax.bar(['Unscaled', 'Scaled'], comparison_df[metric],
           color=['lightblue', 'lightcoral'], alpha=0.7)
    ax.set_ylabel('R2 Score')
    ax.set_title(title)
    ax.grid(True, alpha=0.3)
s
for j, v in enumerate(comparison_df[metric]):
    ax.text(j, v + 0.01, f'{v:.3f}', ha='center', va='bottom')

```

```
plt.tight_layout()
```

```
plt.show()
```

```
scaled_avg_r2 = (test_metrics_scaled['r2_abw'] + test_metrics_scaled['r2_brix']) / 2
```

```
unscaled_avg_r2 = (test_metrics_unscaled['r2_abw'] + test_metrics_unscaled['r2_brix']) / 2
```

```
print(f"Average Test R2 - Unscaled: {unscaled_avg_r2:.4f}")
```

```
print(f"Average Test R2 - Scaled: {scaled_avg_r2:.4f}")
```