```
!pip -q install pandas numpy matplotlib scikit-learn shap lime
```

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


from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.inspection import permutation_importance, PartialDependenceDisplay


SEED = 42
np.random.seed(SEED)
```

```
# Run, A (mg), B (min), Y1 (solubility), Y2 (%CDR)
data = [
(1, 600.0000, 7.00000, 52.79, 79.56),
(2, 400.0000, 18.50000, 47.37, 71.48),
(3, 400.0000, 18.50000, 42.23, 67.37),
(4, 600.0000, 30.00000, 65.86, 85.93),
(5, 400.0000, 18.50000, 46.70, 76.48),
(6, 200.0000, 7.00000, 21.50, 33.89),
(7, 400.0000, 34.76350, 52.28, 79.33),
(8, 400.0000, 18.50000, 37.15, 62.15),
(9, 200.0000, 30.00000, 35.64, 57.03),
(10, 400.0000, 18.50000, 41.56, 66.46),
(11, 682.8430, 18.50000, 74.63, 95.73),
(12, 117.1570, 18.50000, 17.31, 26.08),
(13, 400.0000, 2.23654, 9.83, 25.31),
]


df = pd.DataFrame(data, columns=["Run","A","B","Y1","Y2"])
display(df)
print(df.describe())
```

	Run	A	B	Y1	Y2
0	1	600.000	7.00000	52.79	79.56
1	2	400.000	18.50000	47.37	71.48
2	3	400.000	18.50000	42.23	67.37
3	4	600.000	30.00000	65.86	85.93
4	5	400.000	18.50000	46.70	76.48
5	6	200.000	7.00000	21.50	33.89
6	7	400.000	34.76350	52.28	79.33
7	8	400.000	18.50000	37.15	62.15
8	9	200.000	30.00000	35.64	57.03
9	10	400.000	18.50000	41.56	66.46
10	11	682.843	18.50000	74.63	95.73
11	12	117.157	18.50000	17.31	26.08
12	13	400.000	2.23654	9.83	25.31

```
             Run           A          B         Y1         Y2
count  13.00000   13.000000  13.000000  13.000000  13.000000
mean    7.00000  400.000000  18.500003  41.911538  63.600000
std     3.89444  163.299399   9.389718  18.339206  22.519117
min     1.00000  117.157000   2.236540   9.830000  25.310000
25%     4.00000  400.000000  18.500000  35.640000  57.030000
50%     7.00000  400.000000  18.500000  42.230000  67.370000
75%    10.00000  400.000000  18.500000  52.280000  79.330000
max    13.00000  682.843000  34.763500  74.630000  95.730000
```

```
import numpy as np

# Validation points: R1, R9, Optimized
val_points = [(600.0, 7.0), (200.0, 30.0), (600.0, 30.0)]

def is_val_point(a, b):
    return any(np.isclose(a, aa) and np.isclose(b, bb) for aa, bb in val_points)

# df must already exist with columns: A, B, Y1, Y2
mask_val = df.apply(lambda r: is_val_point(r["A"], r["B"]), axis=1)

train_df = df.loc[~mask_val].copy()
val_df   = df.loc[ mask_val].copy()

print("Training n =", len(train_df), " Validation n =", len(val_df))
display(train_df)
display(val_df)
```

```
Training n = 10  Validation n = 3
```

	Run	A	B	Y1	Y2
1	2	400.000	18.50000	47.37	71.48
2	3	400.000	18.50000	42.23	67.37
4	5	400.000	18.50000	46.70	76.48
5	6	200.000	7.00000	21.50	33.89
6	7	400.000	34.76350	52.28	79.33
7	8	400.000	18.50000	37.15	62.15
9	10	400.000	18.50000	41.56	66.46
10	11	682.843	18.50000	74.63	95.73
11	12	117.157	18.50000	17.31	26.08
12	13	400.000	2.23654	9.83	25.31

	Run	A	B	Y1	Y2
0	1	600.0	7.0	52.79	79.56
3	4	600.0	30.0	65.86	85.93
8	9	200.0	30.0	35.64	57.03

```
X_train = train_df[["A","B"]]
y1_train = train_df["Y1"].values
y2_train = train_df["Y2"].values


rf_y1 = RandomForestRegressor(n_estimators=800, random_state=SEED)
rf_y2 = RandomForestRegressor(n_estimators=800, random_state=SEED)


rf_y1.fit(X_train, y1_train)
rf_y2.fit(X_train, y2_train)


print("RF models trained.")
```

```
RF models trained.
```

```
import numpy as np
import pandas as pd
from copy import deepcopy
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

def loocv_predict(model, X, y):
    loo = LeaveOneOut()
    preds = np.zeros(len(y), dtype=float)

    for tr, te in loo.split(X):
        m = deepcopy(model)
        m.fit(X.iloc[tr], y[tr])
        preds[te[0]] = m.predict(X.iloc[te])[0]

    return preds

def report_metrics(y_true, y_pred, label):
    r2 = r2_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)   # no squared=False
    rmse = float(np.sqrt(mse))                 # RMSE = sqrt(MSE)

    print(f"{label}: R2={r2:.4f}  MAE={mae:.4f}  RMSE={rmse:.4f}")
    return r2, mae, rmse

# ---- Run LOOCV predictions (RF models must already be trained) ----
pred_loocv_y1 = loocv_predict(rf_y1, X_train, y1_train)
pred_loocv_y2 = loocv_predict(rf_y2, X_train, y2_train)

m_y1 = report_metrics(y1_train, pred_loocv_y1, "RF (LOOCV) Y1")
m_y2 = report_metrics(y2_train, pred_loocv_y2, "RF (LOOCV) Y2")

metrics_table = pd.DataFrame([
    {"Model": "RF", "Response": "Y1", "R2": m_y1[0], "MAE": m_y1[1], "RMSE": m_y1[2]},
    {"Model": "RF", "Response": "Y2", "R2": m_y2[0], "MAE": m_y2[1], "RMSE": m_y2[2]},
])

metrics_table.to_csv("Table_RF_LOOCV_train.csv", index=False)
display(metrics_table)
print("Saved: Table_RF_LOOCV_train.csv")
```

```
RF (LOOCV) Y1: R2=0.1651  MAE=12.0030  RMSE=16.4042
RF (LOOCV) Y2: R2=0.2217  MAE=14.4085  RMSE=20.0952
```

	Model	Response	R2	MAE	RMSE
0	RF	Y1	0.165112	12.003022	16.404238
1	RF	Y2	0.221661	14.408530	20.095215

```
Saved: Table_RF_LOOCV_train.csv
```

```
X_val = val_df[["A","B"]]
val_pred_y1 = rf_y1.predict(X_val)
val_pred_y2 = rf_y2.predict(X_val)


# QbD predicted values (Table 4) for comparison
qbd_pred = {
(600.0, 7.0): (48.82, 71.99),
(200.0, 30.0): (34.99, 55.20),
(600.0, 30.0): (70.64, 98.47),
}


val_table = val_df[["Run","A","B","Y1","Y2"]].copy()
val_table["RF_Y1_pred"] = val_pred_y1
val_table["RF_Y2_pred"] = val_pred_y2


val_table["RF_Y1_%error"] = 100*(val_table["RF_Y1_pred"]-val_table["Y1"])/val_table["Y1"]
val_table["RF_Y2_%error"] = 100*(val_table["RF_Y2_pred"]-val_table["Y2"])/val_table["Y2"]


val_table["QbD_Y1_pred"] = val_table.apply(lambda r: qbd_pred[(float(r["A"]), float(r["B"]))][0], axis=1)
val_table["QbD_Y2_pred"] = val_table.apply(lambda r: qbd_pred[(float(r["A"]), float(r["B"]))][1], axis=1)


val_table["QbD_Y1_%error"] = 100*(val_table["QbD_Y1_pred"]-val_table["Y1"])/val_table["Y1"]
val_table["QbD_Y2_%error"] = 100*(val_table["QbD_Y2_pred"]-val_table["Y2"])/val_table["Y2"]


val_table.to_csv("Table_RF_validation_checkpoints.csv", index=False)
display(val_table)
print("Saved: Table_RF_validation_checkpoints.csv")
```

	Run	A	B	Y1	Y2	RF_Y1_pred	RF_Y2_pred	RF_Y1_%error	RF_Y2_%error	QbD_Y1_pred	QbD_Y2_pred	QbD_Y1_%error	QbD_Y2_%error
0	1	600.0	7.0	52.79	79.56	46.676787	51.429836	-11.580247	-35.357169	48.82	71.99	-7.520364	-9.514832
3	4	600.0	30.0	65.86	85.93	66.092435	89.079350	0.352923	3.665018	70.64	98.47	7.257820	14.593274
8	9	200.0	30.0	35.64	57.03	28.078118	42.132041	-21.217401	-26.123022	34.99	55.20	-1.823793	-3.208837

```
Saved: Table_RF_validation_checkpoints.csv
```

```
import matplotlib.pyplot as plt

def pred_vs_obs(y_true, y_pred, title, filename):
    plt.figure()
    plt.scatter(y_true, y_pred)

    mn = float(min(np.min(y_true), np.min(y_pred)))
    mx = float(max(np.max(y_true), np.max(y_pred)))

    plt.plot([mn, mx], [mn, mx])
    plt.xlabel("Observed")
    plt.ylabel("Predicted (LOOCV)")
    plt.title(title)
    plt.tight_layout()
    plt.savefig(filename, dpi=300)
    plt.close()

# ---- Call the function (these lines must NOT be indented) ----
pred_vs_obs(y1_train, pred_loocv_y1, "RF LOOCV: Predicted vs Observed (Y1)", "Fig_PredVsObs_Y1.png")
pred_vs_obs(y2_train, pred_loocv_y2, "RF LOOCV: Predicted vs Observed (Y2)", "Fig_PredVsObs_Y2.png")

print("Saved: Fig_PredVsObs_Y1.png and Fig_PredVsObs_Y2.png")
```

```
Saved: Fig_PredVsObs_Y1.png and Fig_PredVsObs_Y2.png
```

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

# Use TRAIN set for explanations (recommended)
# Make sure these exist before running:
# df, X_train, y1_train, y2_train, rf_y1, rf_y2

# ---------------- Y1 ----------------
explainer_y1 = shap.TreeExplainer(rf_y1)
shap_exp_y1 = explainer_y1(X_train)  # Explanation object

# Summary plot (beeswarm)
shap.summary_plot(shap_exp_y1.values, X_train, show=False)
plt.tight_layout()
plt.savefig("Fig_SHAP_summary_Y1.png", dpi=300)
plt.close()

# Dependence plots (A and B)
for feat in ["A", "B"]:
    shap.dependence_plot(feat, shap_exp_y1.values, X_train, show=False)
    plt.tight_layout()
    plt.savefig(f"Fig_SHAP_dependence_Y1_{feat}.png", dpi=300)
    plt.close()

# Local SHAP waterfall: highest and lowest observed Y1 within TRAIN
idx_hi = int(np.argmax(y1_train))
idx_lo = int(np.argmin(y1_train))
```

```
for name, idx in [("high", idx_hi), ("low", idx_lo)]:
    shap.plots.waterfall(shap_exp_y1[idx], show=False)
    plt.tight_layout()
    plt.savefig(f"Fig_SHAP_waterfall_Y1_{name}.png", dpi=300)
    plt.close()

print("Saved SHAP figures for Y1")

# ---------------- Y2 ----------------
explainer_y2 = shap.TreeExplainer(rf_y2)
shap_exp_y2 = explainer_y2(X_train)

shap.summary_plot(shap_exp_y2.values, X_train, show=False)
plt.tight_layout()
plt.savefig("Fig_SHAP_summary_Y2.png", dpi=300)
plt.close()

for feat in ["A", "B"]:
    shap.dependence_plot(feat, shap_exp_y2.values, X_train, show=False)
    plt.tight_layout()
    plt.savefig(f"Fig_SHAP_dependence_Y2_{feat}.png", dpi=300)
    plt.close()

idx_hi2 = int(np.argmax(y2_train))
idx_lo2 = int(np.argmin(y2_train))

for name, idx in [("high", idx_hi2), ("low", idx_lo2)]:
    shap.plots.waterfall(shap_exp_y2[idx], show=False)
    plt.tight_layout()
    plt.savefig(f"Fig_SHAP_waterfall_Y2_{name}.png", dpi=300)
    plt.close()

print("Saved SHAP figures for Y2")

# ---------------- SHAP global importance table ----------------
imp_y1 = np.abs(shap_exp_y1.values).mean(axis=0)
imp_y2 = np.abs(shap_exp_y2.values).mean(axis=0)

shap_table = pd.DataFrame({
    "Feature": list(X_train.columns),
    "mean_abs_SHAP_Y1": imp_y1,
    "mean_abs_SHAP_Y2": imp_y2
})

shap_table.to_csv("Table_SHAP_importance.csv", index=False)
display(shap_table)
print("Saved: Table_SHAP_importance.csv")
```

```
/tmp/ipython-input-1010726122.py:15: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`. In a future version this function will no longer use th
  shap.summary_plot(shap_exp_y1.values, X_train, show=False)
Saved SHAP figures for Y1
/tmp/ipython-input-1010726122.py:43: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`. In a future version this function will no longer use th
  shap.summary_plot(shap_exp_y2.values, X_train, show=False)
Saved SHAP figures for Y2
```

	Feature	mean_abs_SHAP_Y1	mean_abs_SHAP_Y2
0	A	5.219187	8.014233
1	B	5.724004	8.042018

```
Saved: Table_SHAP_importance.csv
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.inspection import permutation_importance

# SEED must exist; if not, define it
try:
    SEED
except NameError:
    SEED = 42

def run_pfi(model, X, y, title, out_csv, out_png):
    r = permutation_importance(
        model, X, y,
        scoring="neg_mean_absolute_error",
        n_repeats=200,
        random_state=SEED
    )

    pfi = pd.DataFrame({
        "Feature": list(X.columns),
        "Importance_mean": r.importances_mean,
        "Importance_std": r.importances_std
    }).sort_values("Importance_mean", ascending=False)

    pfi.to_csv(out_csv, index=False)

    plt.figure()
    plt.bar(pfi["Feature"], pfi["Importance_mean"], yerr=pfi["Importance_std"])
    plt.ylabel("PFI (increase in MAE units)")
    plt.title(title)
    plt.tight_layout()
    plt.savefig(out_png, dpi=300)
    plt.close()
```

```
    return pfi

# ---- Run PFI on TRAIN set ----
pfi_y1 = run_pfi(rf_y1, X_train, y1_train, "Permutation Feature Importance (RF–Y1)", "Table_PFI_Y1.csv", "Fig_PFI_Y1.png")
pfi_y2 = run_pfi(rf_y2, X_train, y2_train, "Permutation Feature Importance (RF–Y2)", "Table_PFI_Y2.csv", "Fig_PFI_Y2.png")

display(pfi_y1)
display(pfi_y2)
print("Saved: Table_PFI_Y1.csv, Fig_PFI_Y1.png, Table_PFI_Y2.csv, Fig_PFI_Y2.png")
```

	Feature	Importance_mean	Importance_std
0	A	6.928346	2.151580
1	B	5.020840	1.816498

	Feature	Importance_mean	Importance_std
0	A	8.846882	3.002529
1	B	6.997089	2.700590

```
Saved: Table_PFI_Y1.csv, Fig_PFI_Y1.png, Table_PFI_Y2.csv, Fig_PFI_Y2.png
```

```
# 1D PDP
fig, ax = plt.subplots(1, 2, figsize=(10,4))
PartialDependenceDisplay.from_estimator(rf_y1, X_train, ["A"], ax=ax[0], grid_resolution=50)
PartialDependenceDisplay.from_estimator(rf_y1, X_train, ["B"], ax=ax[1], grid_resolution=50)
plt.suptitle("PDP (1D) — RF–Y1")
plt.tight_layout()
plt.savefig("Fig_PDP1D_Y1.png", dpi=300)
plt.close()


fig, ax = plt.subplots(1, 2, figsize=(10,4))
PartialDependenceDisplay.from_estimator(rf_y2, X_train, ["A"], ax=ax[0], grid_resolution=50)
PartialDependenceDisplay.from_estimator(rf_y2, X_train, ["B"], ax=ax[1], grid_resolution=50)
plt.suptitle("PDP (1D) — RF–Y2")
plt.tight_layout()
plt.savefig("Fig_PDP1D_Y2.png", dpi=300)
plt.close()


# 2D PDP
fig, ax = plt.subplots(figsize=(6,5))
PartialDependenceDisplay.from_estimator(rf_y1, X_train, [("A","B")], ax=ax, grid_resolution=40)
plt.title("PDP (2D A×B) — RF–Y1")
plt.tight_layout()
plt.savefig("Fig_PDP2D_Y1.png", dpi=300)
plt.close()


fig, ax = plt.subplots(figsize=(6,5))
PartialDependenceDisplay.from_estimator(rf_y2, X_train, [("A","B")], ax=ax, grid_resolution=40)
plt.title("PDP (2D A×B) — RF–Y2")
plt.tight_layout()
plt.savefig("Fig_PDP2D_Y2.png", dpi=300)
plt.close()


# ICE
fig, ax = plt.subplots(figsize=(6,4))
PartialDependenceDisplay.from_estimator(rf_y1, X_train, ["A"], kind="individual", ax=ax, grid_resolution=50)
plt.title("ICE — A on RF–Y1")
plt.tight_layout()
plt.savefig("Fig_ICE_Y1_A.png", dpi=300)
plt.close()


fig, ax = plt.subplots(figsize=(6,4))
PartialDependenceDisplay.from_estimator(rf_y1, X_train, ["B"], kind="individual", ax=ax, grid_resolution=50)
plt.title("ICE — B on RF–Y1")
plt.tight_layout()
plt.savefig("Fig_ICE_Y1_B.png", dpi=300)
plt.close()


fig, ax = plt.subplots(figsize=(6,4))
PartialDependenceDisplay.from_estimator(rf_y2, X_train, ["A"], kind="individual", ax=ax, grid_resolution=50)
plt.title("ICE — A on RF–Y2")
plt.tight_layout()
plt.savefig("Fig_ICE_Y2_A.png", dpi=300)
plt.close()


fig, ax = plt.subplots(figsize=(6,4))
PartialDependenceDisplay.from_estimator(rf_y2, X_train, ["B"], kind="individual", ax=ax, grid_resolution=50)
plt.title("ICE — B on RF–Y2")
plt.tight_layout()
plt.savefig("Fig_ICE_Y2_B.png", dpi=300)
plt.close()


print("Saved PDP/ICE figures")
```

```
Saved PDP/ICE figures
```

```
!pip -q install lime
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 12.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for lime (setup.py) ... done
```

```
import numpy as np
import pandas as pd
from lime.lime_tabular import LimeTabularExplainer

# Make sure these exist before running:
# X_train, rf_y1, rf_y2, SEED

feature_names = list(X_train.columns)

lime_explainer = LimeTabularExplainer(
    training_data=X_train.values,
    feature_names=feature_names,
    mode="regression",
    discretize_continuous=True,
    random_state=SEED
)

# Wrapper functions (must return a 1D array of predictions)
def predict_y1(arr):
    arr_df = pd.DataFrame(arr, columns=feature_names)
    return rf_y1.predict(arr_df)

def predict_y2(arr):
    arr_df = pd.DataFrame(arr, columns=feature_names)
    return rf_y2.predict(arr_df)

# Choose one "best" and one "worst" point based on RF predictions (within TRAIN)
pred_train_y1 = rf_y1.predict(X_train)
idx_best = int(np.argmax(pred_train_y1))
idx_worst = int(np.argmin(pred_train_y1))

NUM_SAMPLES = 5000

for label, idx in [("best", idx_best), ("worst", idx_worst)]:
    # Explain Y1
    exp1 = lime_explainer.explain_instance(
        data_row=X_train.values[idx],
        predict_fn=predict_y1,
        num_features=2,
        num_samples=NUM_SAMPLES
    )
    exp1.save_to_file(f"LIME_Y1_{label}.html")

    # Explain Y2 at the same point (so it's comparable)
    exp2 = lime_explainer.explain_instance(
        data_row=X_train.values[idx],
        predict_fn=predict_y2,
        num_features=2,
        num_samples=NUM_SAMPLES
    )
    exp2.save_to_file(f"LIME_Y2_{label}.html")

print("Saved: LIME_Y1_best.html, LIME_Y1_worst.html, LIME_Y2_best.html, LIME_Y2_worst.html")
print("Open them from Colab > Files (left panel) > click the .html files.")
```

```
Saved: LIME_Y1_best.html, LIME_Y1_worst.html, LIME_Y2_best.html, LIME_Y2_worst.html
Open them from Colab > Files (left panel) > click the .html files.
```

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

# ---- Dense grid in A and B ----
A_grid = np.linspace(df["A"].min(), df["A"].max(), 120)
B_grid = np.linspace(df["B"].min(), df["B"].max(), 120)
AA, BB = np.meshgrid(A_grid, B_grid)

grid = pd.DataFrame({"A": AA.ravel(), "B": BB.ravel()})
Z1 = rf_y1.predict(grid).reshape(AA.shape)
Z2 = rf_y2.predict(grid).reshape(AA.shape)

# ---- RF contour plots ----
plt.figure()
plt.contourf(AA, BB, Z1, levels=20)
plt.xlabel("A (mg)")
plt.ylabel("B (min)")
plt.title("RF contour — Y1")
plt.tight_layout()
plt.savefig("Fig_RF_contour_Y1.png", dpi=300)
plt.close()

plt.figure()
plt.contourf(AA, BB, Z2, levels=20)
plt.xlabel("A (mg)")
plt.ylabel("B (min)")
plt.title("RF contour — Y2")
plt.tight_layout()
plt.savefig("Fig_RF_contour_Y2.png", dpi=300)
plt.close()
```

```
# ---- Desirability (maximize both) ----
def desirability_max(y, low, high):
    y = np.asarray(y)
    d = np.zeros_like(y, dtype=float)
    d[y >= high] = 1.0
    mid = (y > low) & (y < high)
    d[mid] = (y[mid] - low) / (high - low)
    return d

Y1_low, Y1_high = df["Y1"].min(), df["Y1"].max()
Y2_low, Y2_high = df["Y2"].min(), df["Y2"].max()

d1 = desirability_max(Z1, Y1_low, Y1_high)
d2 = desirability_max(Z2, Y2_low, Y2_high)
D = np.sqrt(d1 * d2)

# ---- Find optimum ----
best_idx = np.unravel_index(np.argmax(D), D.shape)
A_star = float(AA[best_idx])
B_star = float(BB[best_idx])
Y1_star = float(Z1[best_idx])
Y2_star = float(Z2[best_idx])
D_star = float(D[best_idx])

opt = pd.DataFrame([{
    "A_opt_mg": A_star,
    "B_opt_min": B_star,
    "Y1_pred": Y1_star,
    "Y2_pred": Y2_star,
    "Overall_desirability": D_star
}])

opt.to_csv("Table_RF_optimum.csv", index=False)
display(opt)

# ---- Desirability contour ----
plt.figure()
plt.contourf(AA, BB, D, levels=20)
plt.scatter([A_star], [B_star])
plt.xlabel("A (mg)")
plt.ylabel("B (min)")
plt.title("Overall desirability (RF)")
plt.tight_layout()
plt.savefig("Fig_Desirability_contour.png", dpi=300)
plt.close()

print("Saved: Fig_RF_contour_Y1.png, Fig_RF_contour_Y2.png, Fig_Desirability_contour.png, Table_RF_optimum.csv")
```

	A_opt_mg	B_opt_min	Y1_pred	Y2_pred	Overall_desirability
0	544.986748	26.836762	66.092435	89.07935	0.886706

```
Saved: Fig_RF_contour_Y1.png, Fig_RF_contour_Y2.png, Fig_Desirability_contour.png, Table_RF_optimum.csv
```

```
import os
from google.colab import files


!zip -r Results_RF_XAI.zip *.png *.csv *.html
files.download("Results_RF_XAI.zip")
```

```
  adding: Fig_Desirability_contour.png (deflated 29%)
  adding: Fig_ICE_Y1_A.png (deflated 14%)
  adding: Fig_ICE_Y1_B.png (deflated 16%)
  adding: Fig_ICE_Y2_A.png (deflated 15%)
  adding: Fig_ICE_Y2_B.png (deflated 17%)
  adding: Fig_PDP1D_Y1.png (deflated 16%)
  adding: Fig_PDP1D_Y2.png (deflated 17%)
  adding: Fig_PDP2D_Y1.png (deflated 9%)
  adding: Fig_PDP2D_Y2.png (deflated 9%)
  adding: Fig_PFI_Y1.png (deflated 30%)
  adding: Fig_PFI_Y2.png (deflated 28%)
  adding: Fig_PredVsObs_Y1.png (deflated 17%)
  adding: Fig_PredVsObs_Y2.png (deflated 17%)
  adding: Fig_RF_contour_Y1.png (deflated 32%)
  adding: Fig_RF_contour_Y2.png (deflated 32%)
  adding: Fig_SHAP_dependence_Y1_A.png (deflated 29%)
  adding: Fig_SHAP_dependence_Y1_B.png (deflated 28%)
  adding: Fig_SHAP_dependence_Y2_A.png (deflated 31%)
  adding: Fig_SHAP_dependence_Y2_B.png (deflated 28%)
  adding: Fig_SHAP_summary_Y1.png (deflated 15%)
  adding: Fig_SHAP_summary_Y2.png (deflated 16%)
  adding: Fig_SHAP_waterfall_Y1_high.png (deflated 19%)
  adding: Fig_SHAP_waterfall_Y1_low.png (deflated 18%)
  adding: Fig_SHAP_waterfall_Y2_high.png (deflated 19%)
  adding: Fig_SHAP_waterfall_Y2_low.png (deflated 19%)
  adding: Table_PFI_Y1.csv (deflated 20%)
  adding: Table_PFI_Y2.csv (deflated 20%)
  adding: Table_RF_LOOCV_train.csv (deflated 24%)
  adding: Table_RF_optimum.csv (deflated 20%)
  adding: Table_RF_validation_checkpoints.csv (deflated 45%)
  adding: Table_SHAP_importance.csv (deflated 19%)
  adding: LIME_Y1_best.html (deflated 80%)
  adding: LIME_Y1_worst.html (deflated 80%)
  adding: LIME_Y2_best.html (deflated 80%)
  adding: LIME_Y2_worst.html (deflated 80%)
```