Q3:

```
#!/usr/bin/env python

from __future__ import annotations

import argparse
from pathlib import Path
from typing import Dict, List, Tuple

import numpy as np
import pandas as pd
from surprise import Dataset, Reader, SVD, SVDpp, KNNBaseline
from surprise.model_selection import GridSearchCV, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import matplotlib.pyplot as plt
import seaborn as sns

# ---------------------------------------------------------------------------
# Helpers
# ---------------------------------------------------------------------------

def load_data(dir_: Path) -> Tuple[pd.DataFrame, pd.DataFrame]:
    ratings = pd.read_csv(dir_ / "train_data_movie_rate.csv")
    test = pd.read_csv(dir_ / "test_data.csv")
    return ratings, test


def clean_ratings(df: pd.DataFrame) -> pd.DataFrame:
    df = df.dropna(subset=["user_id", "item_id", "label"]).copy()
    df["label"] = df["label"].astype(float)
    return df.groupby(["user_id", "item_id"], as_index=False)["label"].mean()


def build_dataset(ratings: pd.DataFrame) -> Dataset:
    reader = Reader(rating_scale=(ratings.label.min(), ratings.label.max()))
    return Dataset.load_from_df(ratings[["user_id", "item_id", "label"]], reader)

# ---------------------------------------------------------------------------
# CV tuning helper
# ---------------------------------------------------------------------------
```

```
def tune_algo(algo_cls, grid: Dict, data: Dataset, name: str):
    gs = GridSearchCV(algo_cls, grid, measures=["rmse"], cv=3, n_jobs=-1)
    gs.fit(data)
    print(f"✓ {name} best RMSE {gs.best_score['rmse']:.4f} params {gs.best_params['rmse']}")
    model = algo_cls(**gs.best_params["rmse"])
    model.fit(data.build_full_trainset())
    return model, gs.best_params["rmse"], gs.best_score["rmse"]

# ---------------------------------------------------------------------------
# Metric helper
# ---------------------------------------------------------------------------

def metric_dict(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    return {
        "RMSE": mse ** 0.5,
        "MSE": mse,
        "MAE": mean_absolute_error(y_true, y_pred),
        "R2": r2_score(y_true, y_pred),
    }

# ---------------------------------------------------------------------------
# Main
# ---------------------------------------------------------------------------

def main():
    ap = argparse.ArgumentParser(description="Efficient ensemble recommender v2.1 with plots")
    ap.add_argument("--data_dir", type=Path, default=Path("./dataset/"))
    args = ap.parse_args()

    ratings_df_orig, test_df = load_data(args.data_dir)
    ratings_df = clean_ratings(ratings_df_orig.copy())
    data = build_dataset(ratings_df)
    rmin, rmax = ratings_df.label.min(), ratings_df.label.max()

    print("--- Data Exploration Plots ---")

    plt.figure(figsize=(8, 5))
    sns.histplot(ratings_df['label'], bins=int(rmax - rmin) + 1 if rmax > rmin else 5, kde=False)
    plt.title('Distribution of Movie Ratings')
    plt.xlabel('Rating')
    plt.ylabel('Number of Ratings')
    plt.tight_layout()
    plt.show()
```

```
user_activity = ratings_df.groupby('user_id')['label'].count()
plt.figure(figsize=(10, 6))
sns.histplot(user_activity, bins=50, kde=True)
plt.title('Distribution of Number of Ratings per User')
plt.xlabel('Number of Ratings Given by User')
plt.ylabel('Number of Users')
plt.yscale('log')
plt.tight_layout()
plt.show()

# نمودار توزیع تعداد امتیازات به ازای هر آیتم
item_popularity = ratings_df.groupby('item_id')['label'].count()
plt.figure(figsize=(10, 6))
sns.histplot(item_popularity, bins=50, kde=True)
plt.title('Distribution of Number of Ratings per Item')
plt.xlabel('Number of Ratings Received by Item')
plt.ylabel('Number of Items')
plt.yscale('log')
plt.tight_layout()
plt.show()


item_stats = ratings_df.groupby('item_id')['label'].agg(['mean', 'count']).reset_index()
plt.figure(figsize=(10, 6))
sns.scatterplot(x='count', y='mean', data=item_stats, alpha=0.5, edgecolor=None)
plt.title('Average Item Rating vs. Number of Ratings')
plt.xlabel('Number of Ratings for Item (Popularity)')
plt.ylabel('Average Rating for Item')
if item_stats['count'].nunique() > 1:
    plt.xscale('log')
plt.grid(True)
plt.tight_layout()
plt.show()


# ---------------- Hyper-param grids ---------------------------------
svd_grid = {"n_factors": [120, 160], "lr_all": [0.005, 0.007], "reg_all": [0.02, 0.04], "n_epochs": [60]}
svdpp_grid = {"n_factors": [120], "lr_all": [0.005], "reg_all": [0.03], "n_epochs": [40]}
knn_grid = {"k": [40, 60], "min_k": [3], "sim_options": {"name": ["pearson_baseline"], "user_based'

print("\n--- Hyperparameter Tuning ---")
svd_best, svd_params, svd_cv_score = tune_algo(SVD, svd_grid, data, "SVD")
svdpp_best, svdpp_params, svdpp_cv_score = tune_algo(SVDpp, svdpp_grid, data, "SVD++")
knn_best, knn_params, knn_cv_score = tune_algo(KNNBaseline, knn_grid, data, "KNN-Baseline")
```

```
print("\n--- Base Model Performance (after CV) Plot ---")
model_names_cv = ['SVD', 'SVD++', 'KNN-Baseline']
best_rmses_cv = [svd_cv_score, svdpp_cv_score, knn_cv_score]

plt.figure(figsize=(8, 5))
sns.barplot(x=model_names_cv, y=best_rmses_cv)
plt.title('Best RMSE for Each Base Model (after CV)')
plt.xlabel('Model')
plt.ylabel('CV RMSE')
for index, value in enumerate(best_rmses_cv):
    plt.text(index, value + 0.002, f"{value:.4f}", ha='center')
plt.ylim(min(best_rmses_cv) - 0.02, max(best_rmses_cv) + 0.02)
plt.tight_layout()
plt.show()

# ---------------- Bagging SVDs -------------------------------------
print("\n--- Bagging SVDs ---")
bagging_seeds = [7, 42, 2025]
param_whitelist = ["n_factors", "n_epochs", "biased", "lr_all", "reg_all", "random_state"]
base_svd_params = {k: svd_params[k] for k in param_whitelist if k in svd_params}

svd_bag = []
for i, s in enumerate(bagging_seeds):
    p = base_svd_params.copy()
    p["random_state"] = s
    m = SVD(**p)
    print(f"Fitting SVD bag model {i+1} with seed {s}...")
    m.fit(data.build_full_trainset())
    svd_bag.append(m)

base_models = [svd_best, svdpp_best, knn_best] + svd_bag

base_model_labels_for_plot = [svd_best.__class__.__name__ + "_tuned",
                              svdpp_best.__class__.__name__ + "_tuned",
                              knn_best.__class__.__name__ + "_tuned"] + \
                              [m.__class__.__name__ + f"_bag_{i+1}" for i, m in enumerate(svd_bag)]


# ---------------- Blender training ---------------------------------
print("\n--- Blender Training ---")
trainset, holdout = train_test_split(data, test_size=0.2, random_state=1)
print("Re-fitting base models on the smaller trainset for blender...")
for i, m in enumerate(base_models):
    print(f"Fitting {base_model_labels_for_plot[i]} on smaller trainset...")
```

```
    m.fit(trainset)

y_hold, X_hold_list = [], []
print("Generating predictions from base models on holdout set...")
for uid, iid, r_true in holdout:
    y_hold.append(r_true)
    X_hold_list.append([m.predict(uid, iid).est for m in base_models])
y_hold = np.array(y_hold)
X_hold = np.array(X_hold_list)

print("\n--- Base Model Performance on Holdout Set Plot ---")
base_model_rmses_on_holdout = []
for i, model_name_plot in enumerate(base_model_labels_for_plot):
    model_predictions_on_holdout = X_hold[:, i]
    mse_val = mean_squared_error(y_hold, model_predictions_on_holdout)
    rmse_val = mse_val ** 0.5
    base_model_rmses_on_holdout.append(rmse_val)

plt.figure(figsize=(12, 7))
sns.barplot(x=base_model_labels_for_plot, y=base_model_rmses_on_holdout)
plt.title('RMSE of Base Models on Hold-out Set')
plt.xlabel('Base Model')
plt.ylabel('RMSE on Hold-out')
plt.xticks(rotation=30, ha="right")
for index, value in enumerate(base_model_rmses_on_holdout):
    plt.text(index, value + 0.002, f"{value:.4f}", ha='center')
plt.ylim(min(base_model_rmses_on_holdout) - 0.02, max(base_model_rmses_on_holdout) + 0.02)
plt.tight_layout()
plt.show()



blender = LinearRegression(positive=True, fit_intercept=False)
blender.fit(X_hold, y_hold)
weights = blender.coef_ / blender.coef_.sum()
print("Learned weights by blender:", np.round(weights, 3))

print("\n--- Learned Weights Plot ---")
plt.figure(figsize=(10, 6))
sns.barplot(x=base_model_labels_for_plot, y=weights)
plt.title('Learned Weights for Base Models by Blender')
plt.xlabel('Base Model')
plt.ylabel('Weight')
plt.xticks(rotation=30, ha="right")
for index, value in enumerate(weights):
    plt.text(index, value + 0.002, f"{value:.3f}", ha='center')
```

```
plt.ylim(0, max(weights) + 0.05)
plt.tight_layout()
plt.show()


ens_pred_hold = np.clip(X_hold.dot(weights), rmin, rmax)
holdout_metrics = metric_dict(y_hold, ens_pred_hold)
print("Hold-out ensemble metrics:", holdout_metrics)

print("\n--- Ensemble Performance on Holdout Plots ---")

plt.figure(figsize=(8, 8))
plt.scatter(y_hold, ens_pred_hold, alpha=0.3, edgecolor=None)
plt.plot([rmin, rmax], [rmin, rmax], 'r--', linewidth=2, label='y=x (Perfect Prediction)')
plt.title('Ensemble Predictions vs. True Ratings on Hold-out Set')
plt.xlabel('True Ratings (y_hold)')
plt.ylabel('Ensemble Predicted Ratings (ens_pred_hold)')
plt.axis('equal')
plt.xlim([rmin - 0.1, rmax + 0.1])
plt.ylim([rmin - 0.1, rmax + 0.1])
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()


errors_ensemble = y_hold - ens_pred_hold
plt.figure(figsize=(10, 6))
sns.histplot(errors_ensemble, bins=50, kde=True)
plt.title('Distribution of Prediction Errors (Residuals) for Ensemble Model')
plt.xlabel('Error (True Rating - Predicted Rating)')
plt.ylabel('Frequency')
plt.axvline(errors_ensemble.mean(), color='r', linestyle='dashed', linewidth=1, label=f'Mean Error: {
plt.legend()
plt.tight_layout()
plt.show()


plt.figure(figsize=(10, 6))
plt.scatter(ens_pred_hold, errors_ensemble, alpha=0.3, edgecolor=None)
plt.axhline(0, color='r', linestyle='dashed', linewidth=1, label='Zero Error Line')
plt.title('Residual Plot for Ensemble Model (Errors vs. Predicted Values)')
plt.xlabel('Ensemble Predicted Ratings')
plt.ylabel('Prediction Error (Residuals)')
plt.grid(True)
```

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

    print("\n--- Comparison of Absolute Errors (Box Plot) ---")
    all_errors_df_data = {}
    for i, model_name_plot in enumerate(base_model_labels_for_plot):
        model_predictions_on_holdout = X_hold[:, i]
        all_errors_df_data[model_name_plot] = np.abs(y_hold - model_predictions_on_holdout)
    all_errors_df_data['Ensemble'] = np.abs(errors_ensemble)
    all_errors_df = pd.DataFrame(all_errors_df_data)

    plt.figure(figsize=(14, 8))
    sns.boxplot(data=all_errors_df)
    plt.title('Box Plot of Absolute Prediction Errors on Hold-out Set')
    plt.xlabel('Model')
    plt.ylabel('Absolute Error')
    plt.xticks(rotation=30, ha="right")
    plt.tight_layout()
    plt.show()


    # ---------------- Predict test pairs ------------------------------
    print("\n--- Re-fitting base models on FULL training data before predicting on test set ---")
    full_trainset = data.build_full_trainset()
    for i, m in enumerate(base_models):
        print(f"Fitting {base_model_labels_for_plot[i]} on FULL trainset...")
        m.fit(full_trainset)

    print("\n--- Predicting on Test Set ---")
    ests = []
    for u_test, i_test in test_df[["user_id", "item_id"]].values:
        preds_test = np.array([m.predict(u_test, i_test).est for m in base_models])
        ests.append(float(np.clip(preds_test.dot(weights), rmin, rmax)))

    submission = pd.DataFrame({"id": test_df["id"], "label": ests})
    sub_path = args.data_dir / "submission.csv"
    submission.to_csv(sub_path, index=False)
    print(f"Saved submission to {sub_path.resolve()}")


if __name__ == "__main__":
    main()
```

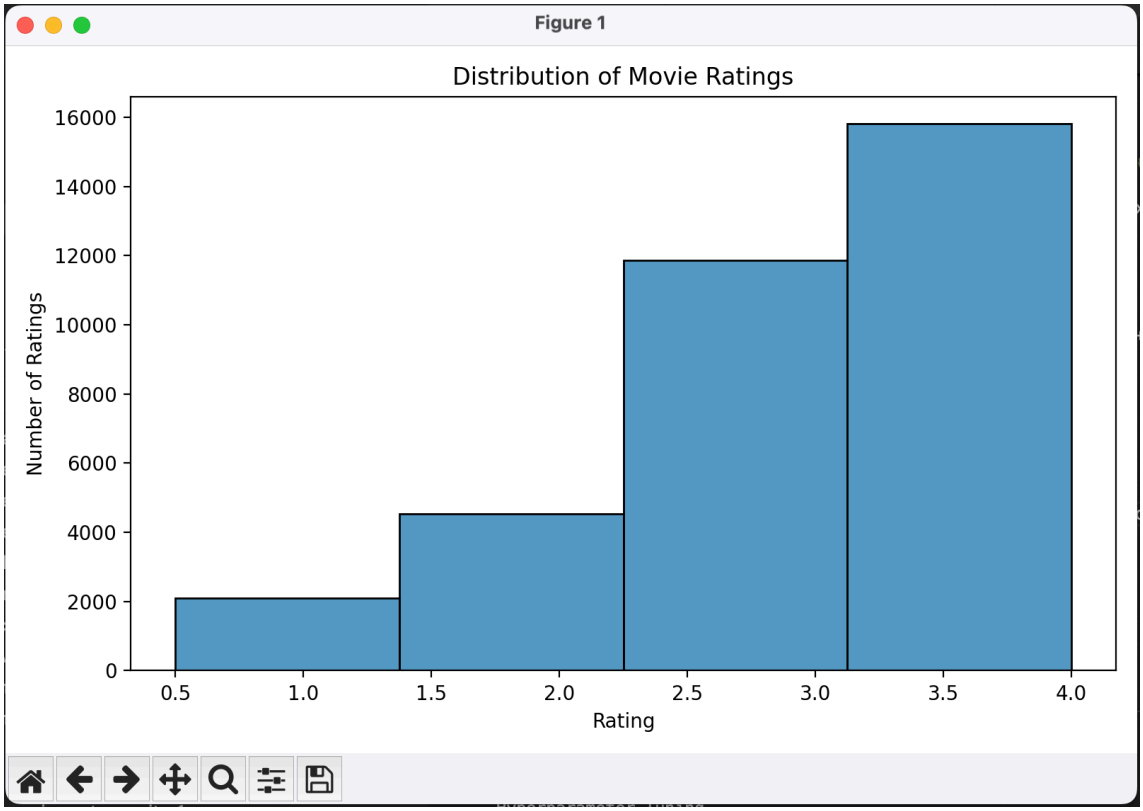


Figure 1
Distribution of Movie Ratings
Number of Ratings
16000
14000
12000
10000
8000
6000
4000
2000
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Rating

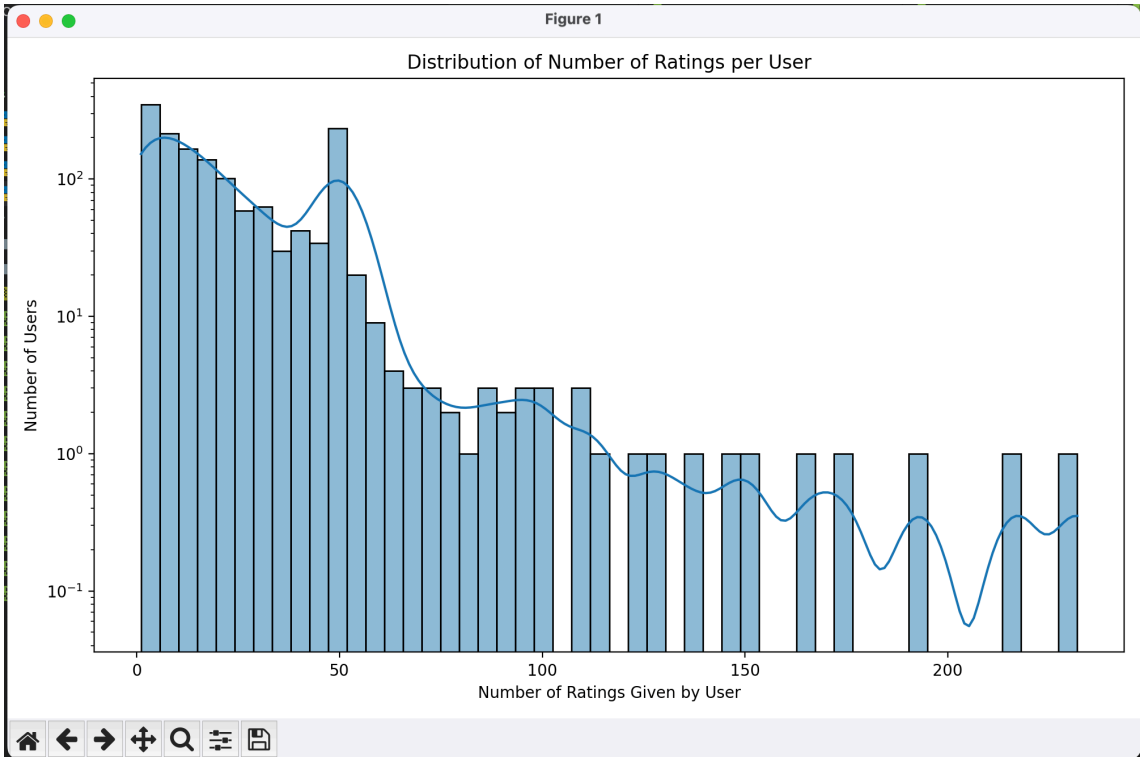


Figure 1
Distribution of Number of Ratings per User
Number of Users
10^2
10^1
10^0
10^{-1}
0
50
100
150
200
Number of Ratings Given by User

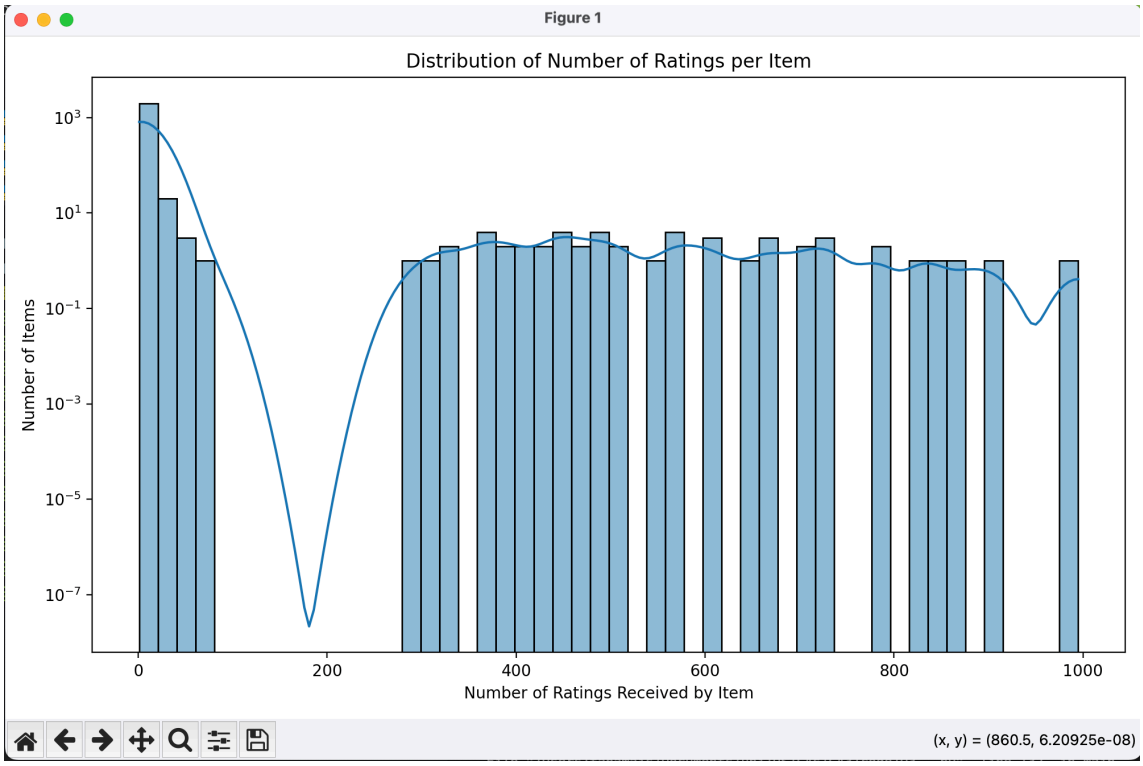


Figure 1
Distribution of Number of Ratings per Item
Number of Items
Number of Ratings Received by Item
(x, y) = (860.5, 6.20925e-08)

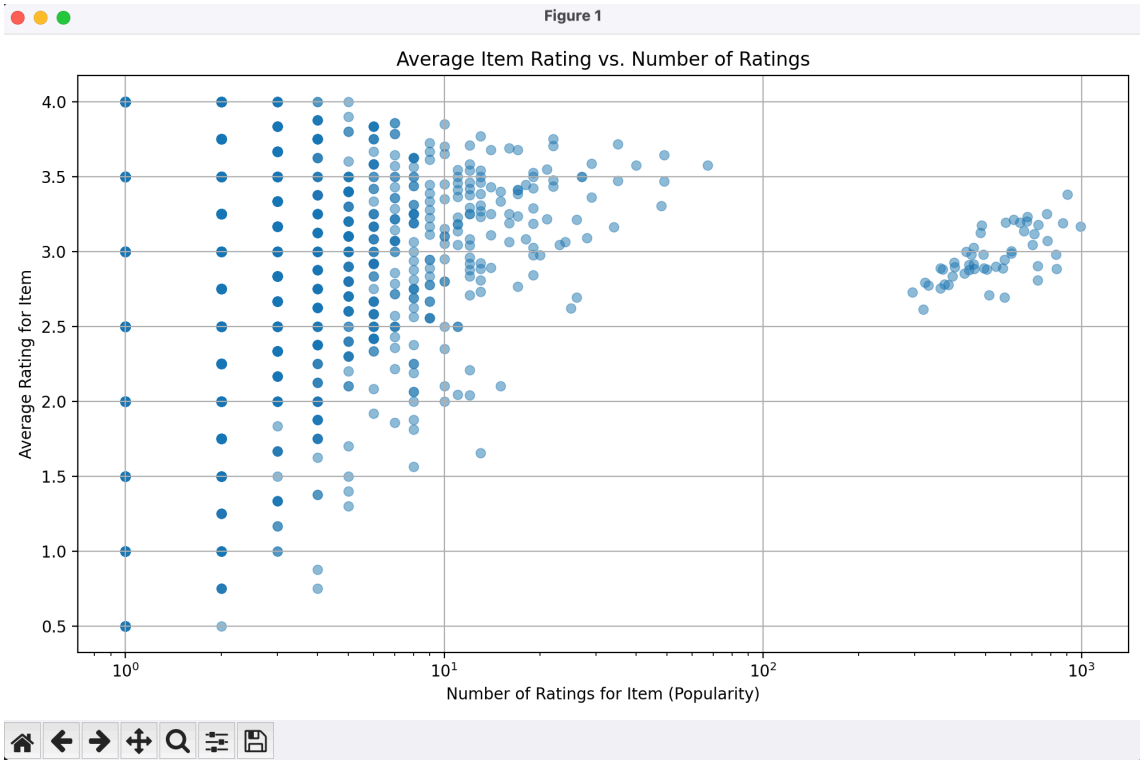


Figure 1
Average Item Rating vs. Number of Ratings
Average Rating for Item
Number of Ratings for Item (Popularity)

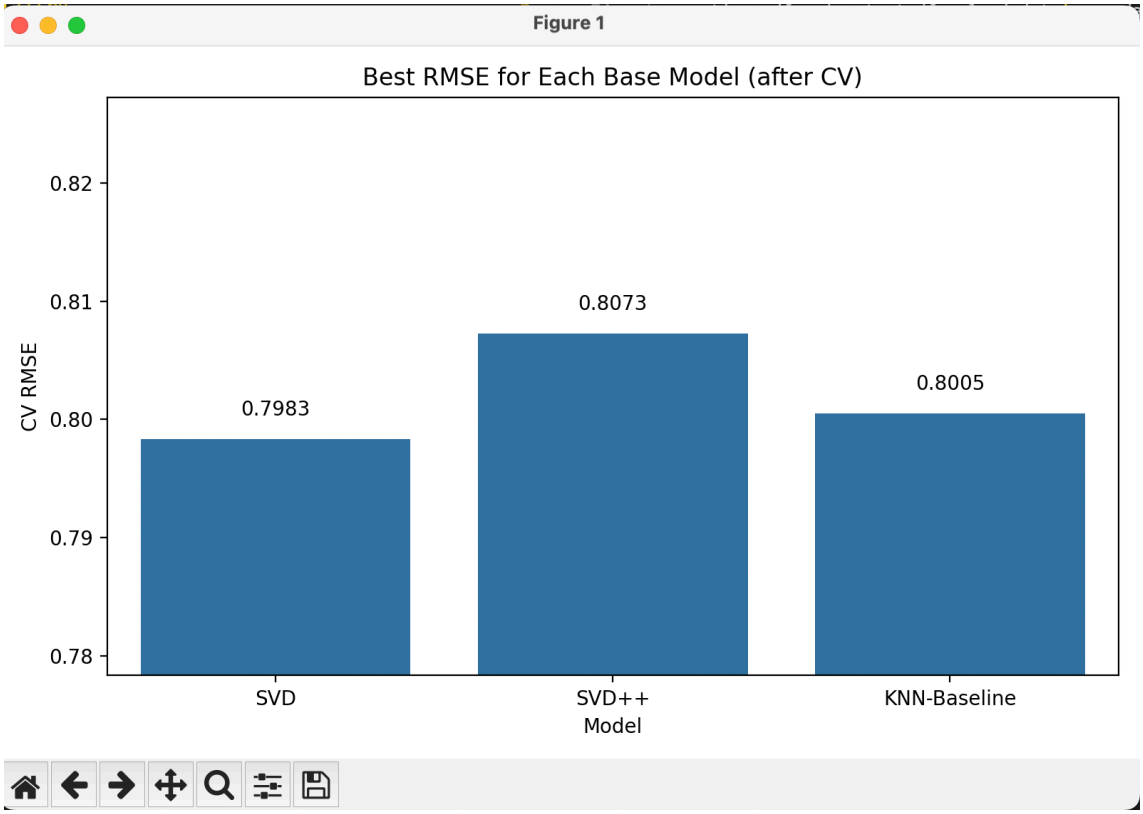


Figure 1
Best RMSE for Each Base Model (after CV)
CV RMSE
0.82
0.81
0.80
0.79
0.78
0.7983
0.8073
0.8005
SVD
SVD++
KNN-Baseline
Model

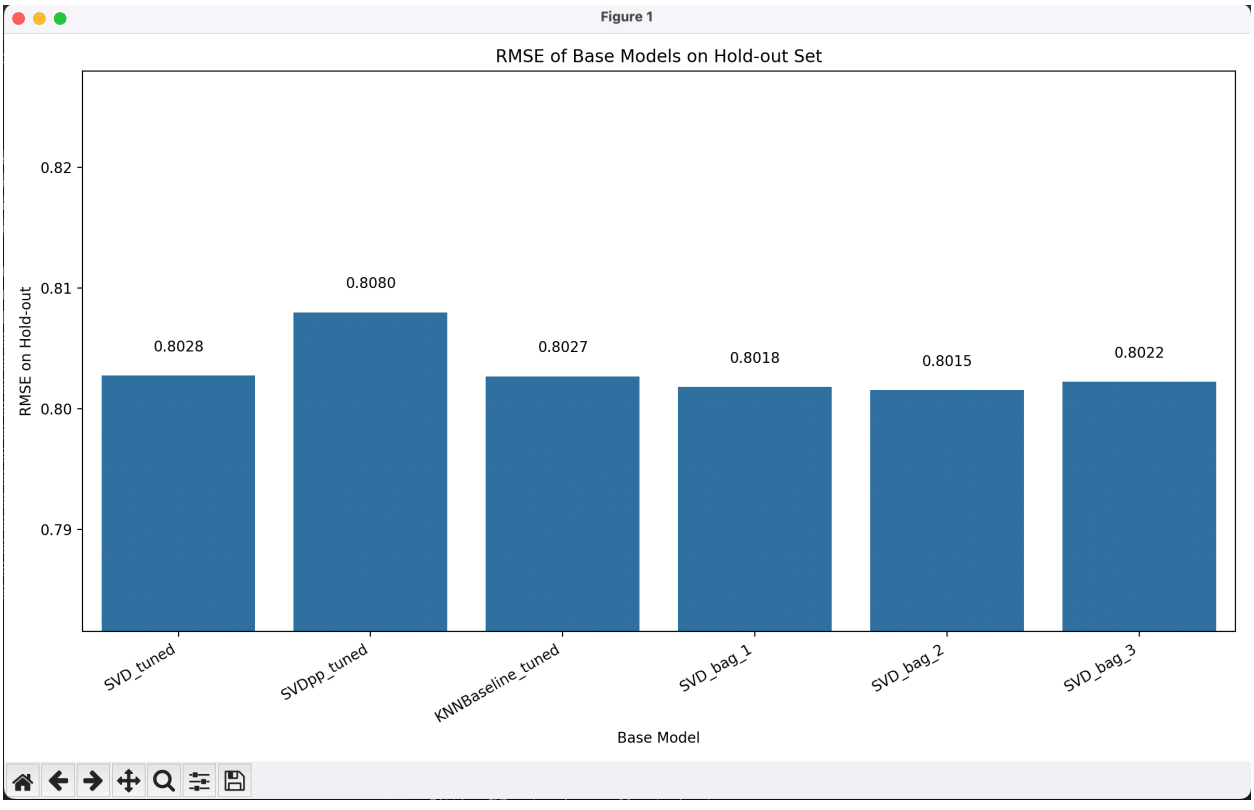


Figure 1
RMSE of Base Models on Hold-out Set
RMSE on Hold-out
0.82
0.81
0.80
0.79
0.8028
0.8080
0.8027
0.8018
0.8015
0.8022
SVD_tuned
SVDpp_tuned
KNNBaseline_tuned
SVD_bag_1
SVD_bag_2
SVD_bag_3
Base Model

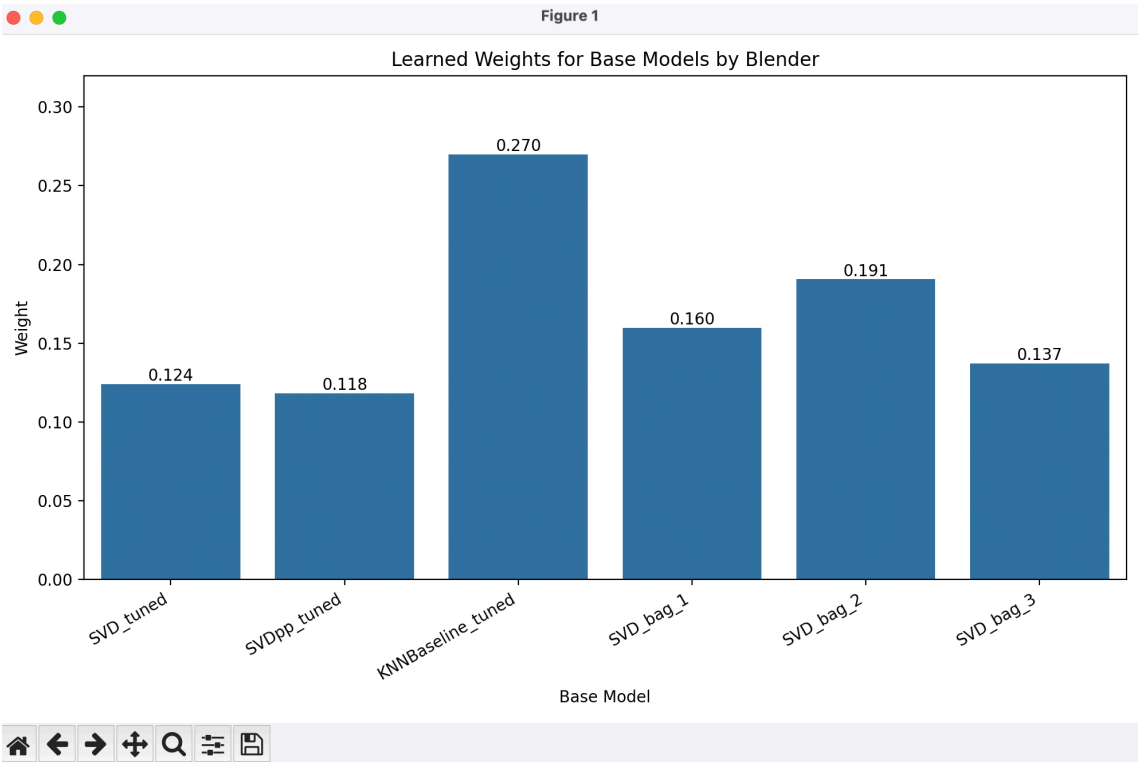


Figure 1
Learned Weights for Base Models by Blender
Weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.124
0.118
0.270
0.160
0.191
0.137
SVD_tuned
SVDpp_tuned
KNNBaseline_tuned
SVD_bag_1
SVD_bag_2
SVD_bag_3
Base Model

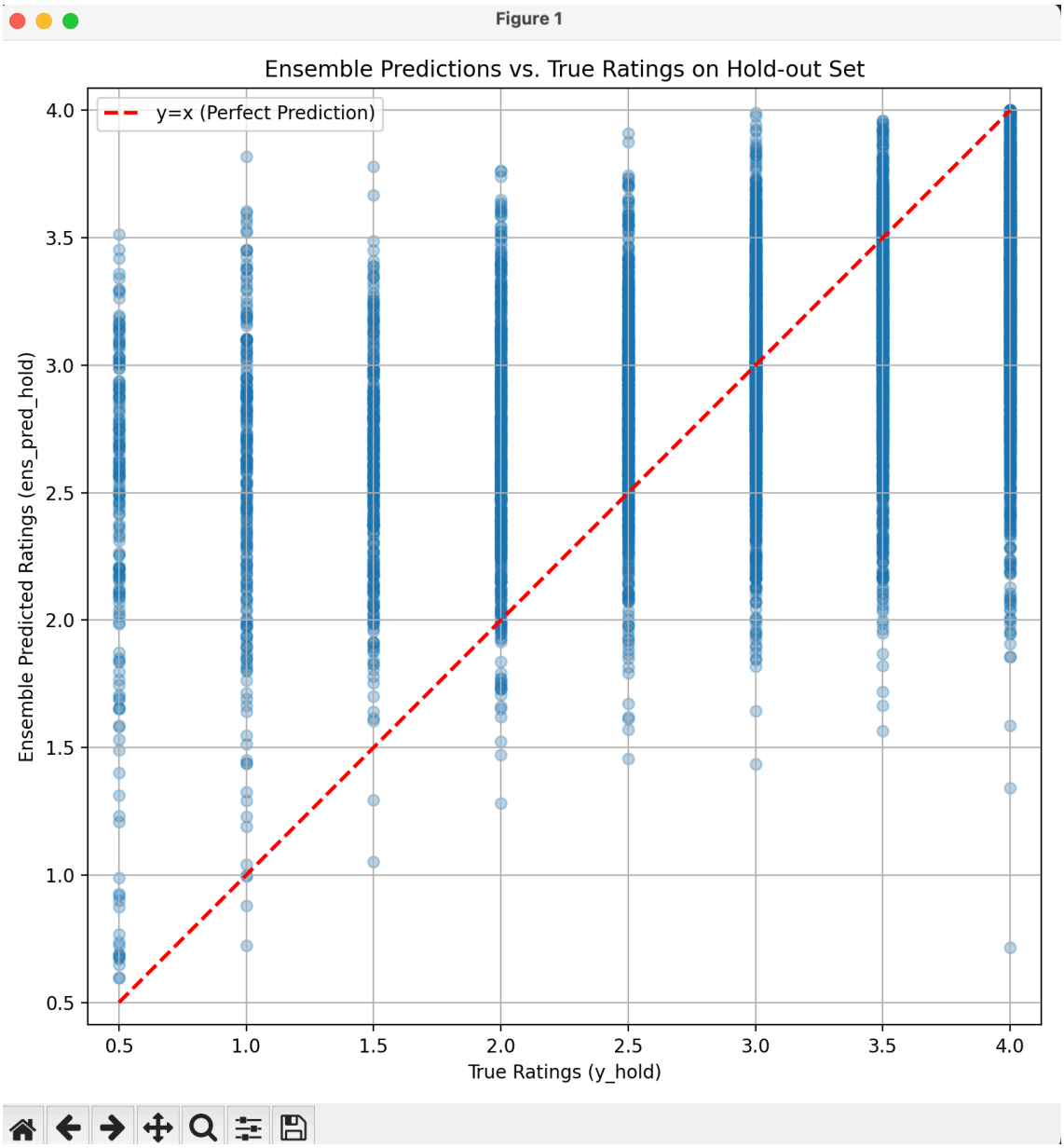


Figure 1
Ensemble Predictions vs. True Ratings on Hold-out Set
y=x (Perfect Prediction)
Ensemble Predicted Ratings (ens_pred_hold)
True Ratings (y_hold)
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

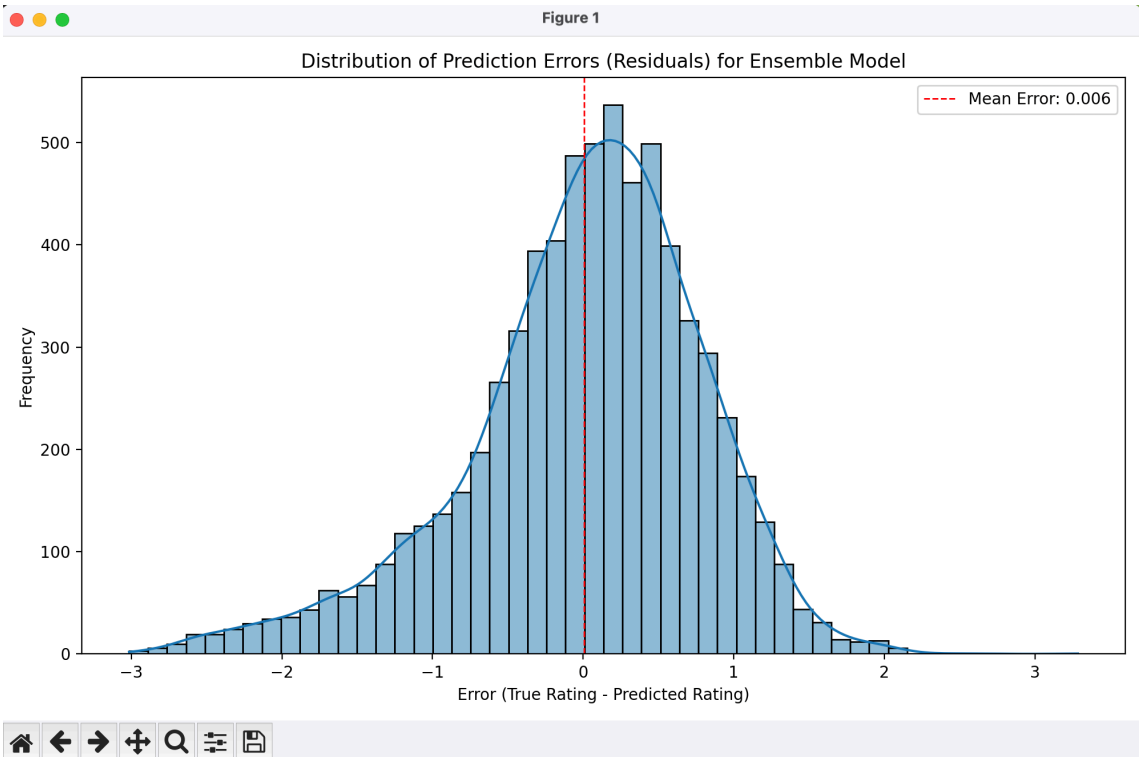


Figure 1
Distribution of Prediction Errors (Residuals) for Ensemble Model
Mean Error: 0.006
Frequency
Error (True Rating - Predicted Rating)

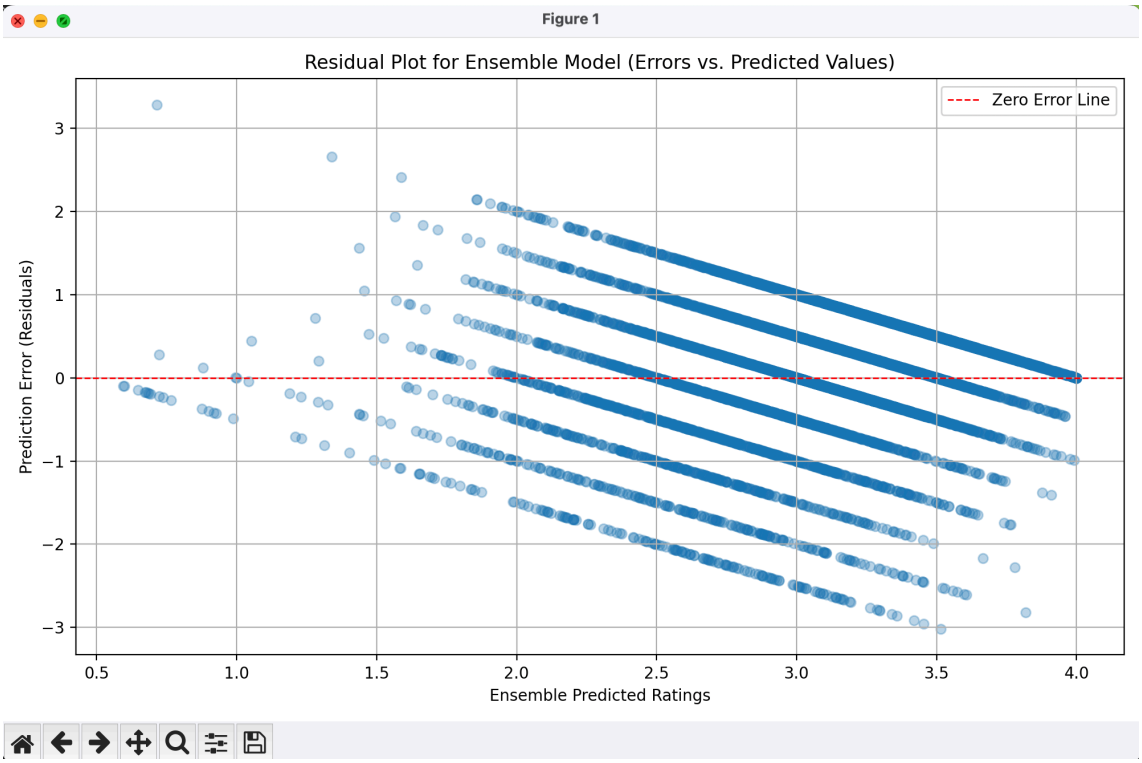
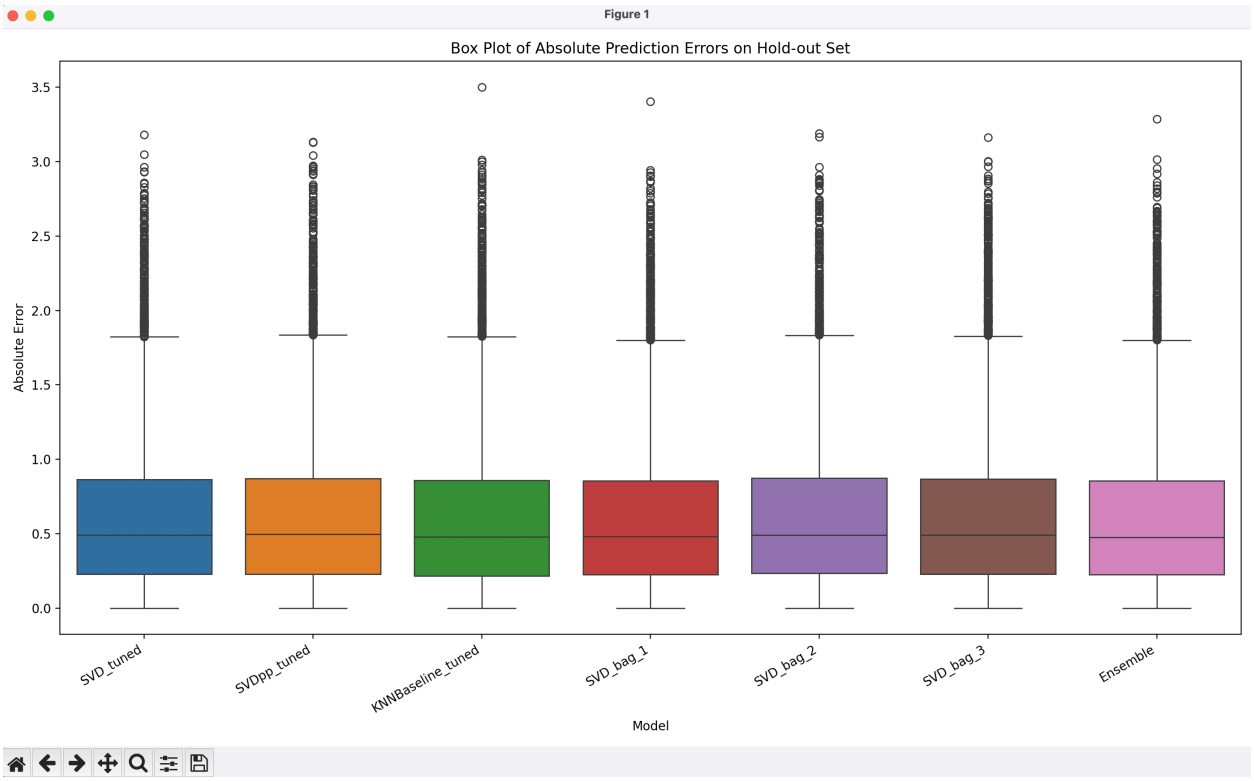


Figure 1
Residual Plot for Ensemble Model (Errors vs. Predicted Values)
Zero Error Line
Prediction Error (Residuals)
Ensemble Predicted Ratings



results:

```
--- Data Exploration Plots ---
2025-05-25 16:43:13.714 python[57743:20631162] +[IMKClient subclass]: chose IMKClient_Legacy
2025-05-25 16:43:13.716 python[57743:20631162] +[IMKInputSession subclass]: chose IMKInputSession_Legacy

--- Hyperparameter Tuning ---
✓ SVD best RMSE 0.8017 params {'n_factors': 160, 'lr_all': 0.005, 'reg_all': 0.04, 'n_epochs': 60}
✓ SVD++ best RMSE 0.8065 params {'n_factors': 120, 'lr_all': 0.005, 'reg_all': 0.03, 'n_epochs': 40}
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Estimating biases using als...
Estimating biases using als...
Done computing similarity matrix.
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the pearson_baseline similarity matrix...
Estimating biases using als...
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
✓ KNN-Baseline best RMSE 0.8014 params {'k': 40, 'min_k': 3, 'sim_options': {'name': 'pearson_baseline', 'user_based': False}}
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

```
--- Base Model Performance (after CV) Plot ---

--- Bagging SVDs ---
Fitting SVD bag model 1 with seed 7...
Fitting SVD bag model 2 with seed 42...
Fitting SVD bag model 3 with seed 2025...

--- Blender Training ---
Re-fitting base models on the smaller trainset for blender...
Fitting SVD_tuned on smaller trainset...
Fitting SVDpp_tuned on smaller trainset...
Fitting KNNBaseline_tuned on smaller trainset...
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Fitting SVD_bag_1 on smaller trainset...
Fitting SVD_bag_2 on smaller trainset...
Fitting SVD_bag_3 on smaller trainset...
Generating predictions from base models on holdout set...

--- Base Model Performance on Holdout Set Plot ---
Learned weights by blender: [0.124 0.118 0.27  0.16  0.191 0.137]

--- Learned Weights Plot ---
Hold-out ensemble metrics: {'RMSE': 0.7898250942038799, 'MSE': 0.6238236794341678, 'MAE': 0.6060043234376087, 'R2': 0.25041683179566376}

--- Ensemble Performance on Holdout Plots ---
Ignoring fixed y limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
```

```
--- Comparison of Absolute Errors (Box Plot) ---

--- Re-fitting base models on FULL training data before predicting on test set ---
Fitting SVD_tuned on FULL trainset...
Fitting SVDpp_tuned on FULL trainset...
Fitting KNNBaseline_tuned on FULL trainset...
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Fitting SVD_bag_1 on FULL trainset...
Fitting SVD_bag_2 on FULL trainset...
Fitting SVD_bag_3 on FULL trainset...

--- Predicting on Test Set ---
Saved submission to /Users/tahamajs/Documents/uni/DS/CAs/CA3/dataset/submission.csv
```