```
import pandas as pd

try:
    df = pd.read_csv('/content/final_dataset.csv')
    display(df.head())
    print(f"Shape of the DataFrame: {df.shape}")
    print("\nData types of each column:")
    display(df.dtypes)
except FileNotFoundError:
    print("Error: '/content/final_dataset.csv' not found.")
except Exception as e:
    print(f"An error occurred: {e}")
```

	fsto	if	ran_l151	ran_l1510	ran_l15100	ran_l1511	ran_l1512	ran_l151
0	0.000000	0.358053	0.715266	0.0	0.0	0.0	0.0	0.
1	0.137494	0.146642	0.000000	0.0	0.0	0.0	0.0	0.
2	0.161976	0.000000	0.000000	0.0	0.0	0.0	0.0	0.
3	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.
4	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.

5 rows × 388 columns

```
Shape of the DataFrame: (376, 388)

Data types of each column:
```

	0
fsto	float64
if	float64
ran_l151	float64
ran_l1510	float64
ran_l15100	float64
...	...
rwz	float64
st	float64
Area	float64
Delay	float64
Power	float64

388 rows × 1 columns

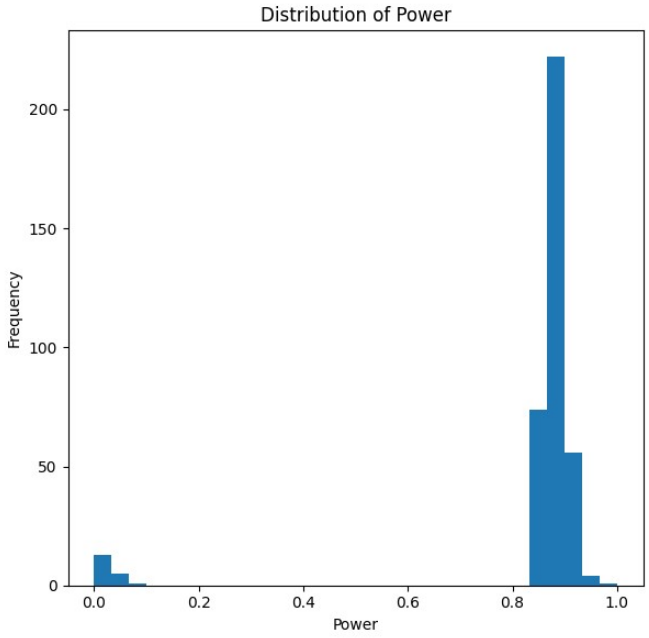
dtype: object

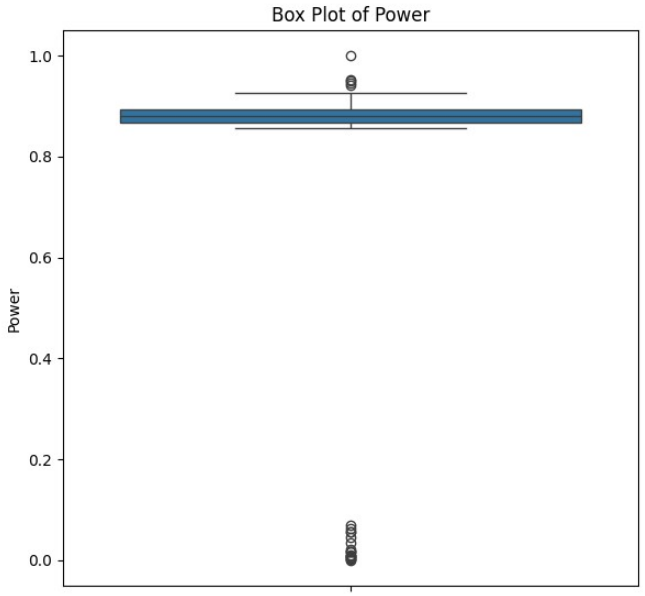
```
import matplotlib.pyplot as plt
import seaborn as sns

# Descriptive statistics for numerical features
display(df.describe())

# Target variable analysis
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(df['Power'], bins=30)
plt.title('Distribution of Power')
plt.xlabel('Power')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sns.boxplot(y=df['Power'])
plt.title('Box Plot of Power')
plt.tight_layout()
plt.show()

# Correlation analysis
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix[['Power', 'Area', 'Delay']].sort_values(by='Power', ascend
plt.title('Correlation Heatmap')
plt.show()

# Missing values
missing_values = df.isnull().sum()
missing_percentage = (missing_values / len(df)) * 100
print("Missing Values Percentage:\n", missing_percentage)

# Outlier detection
plt.figure(figsize=(12, 6))
for i, col in enumerate(['Power', 'Area', 'Delay']):
    plt.subplot(1, 3, i + 1)
    sns.boxplot(y=df[col])
    plt.title(f'Box Plot of {col}')
plt.tight_layout()
plt.show()

# Recipe feature analysis (example: distribution of first 5 recipe features)
plt.figure(figsize=(16, 6))
for i, col in enumerate(df.columns[:5]):
    plt.subplot(1, 5, i + 1)
    plt.hist(df[col], bins=20)
    plt.title(f'Distribution of {col}')
```

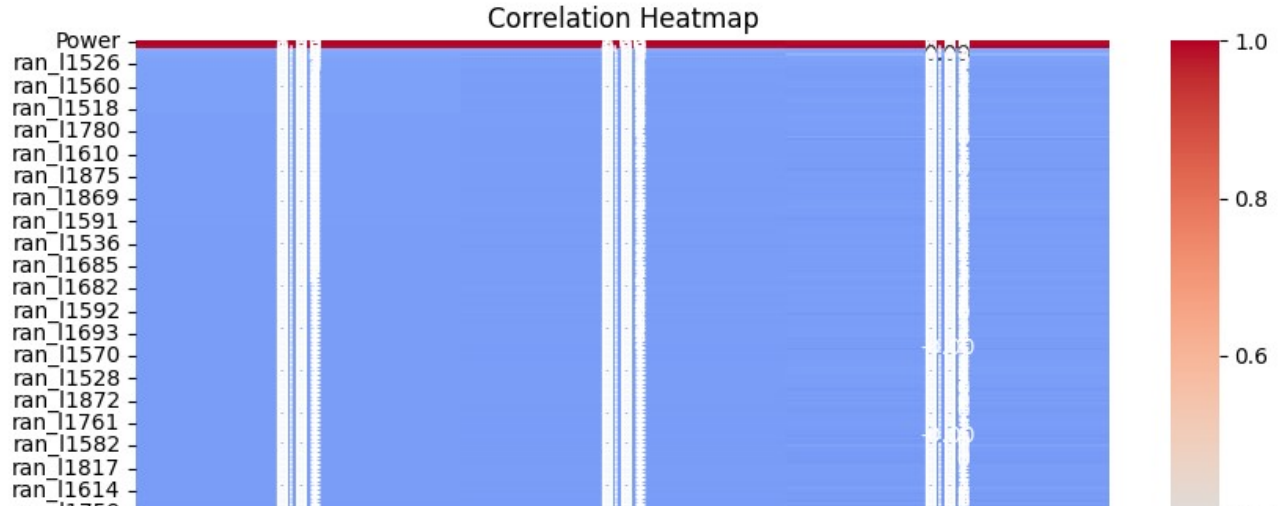
```
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

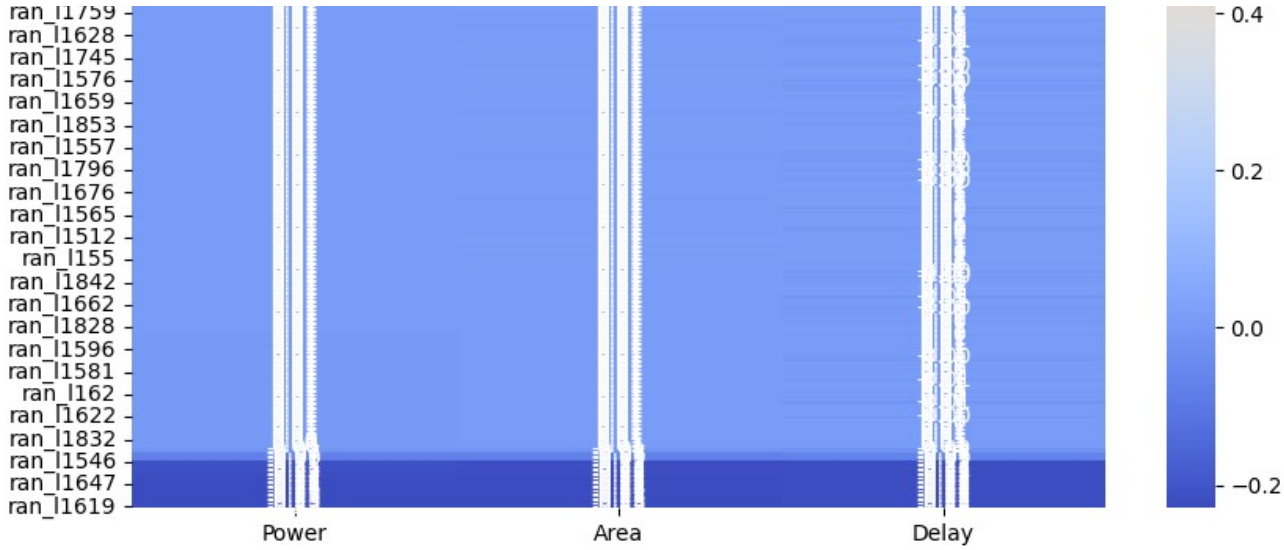
	fsto	if	ran_l151	ran_l1510	ran_l15100	ran_l1511	ran_l151
count	376.000000	376.000000	376.000000	376.000000	376.000000	376.000000	376.0000(
mean	0.123441	0.115378	0.001902	0.001621	0.001386	0.001450	0.0020(
std	0.120208	0.123359	0.036887	0.031426	0.026878	0.028118	0.0399(
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
50%	0.125801	0.128825	0.000000	0.000000	0.000000	0.000000	0.0000(
75%	0.160626	0.158905	0.000000	0.000000	0.000000	0.000000	0.0000(
max	0.484853	0.590113	0.715266	0.609374	0.521189	0.545235	0.77497

8 rows × 388 columns

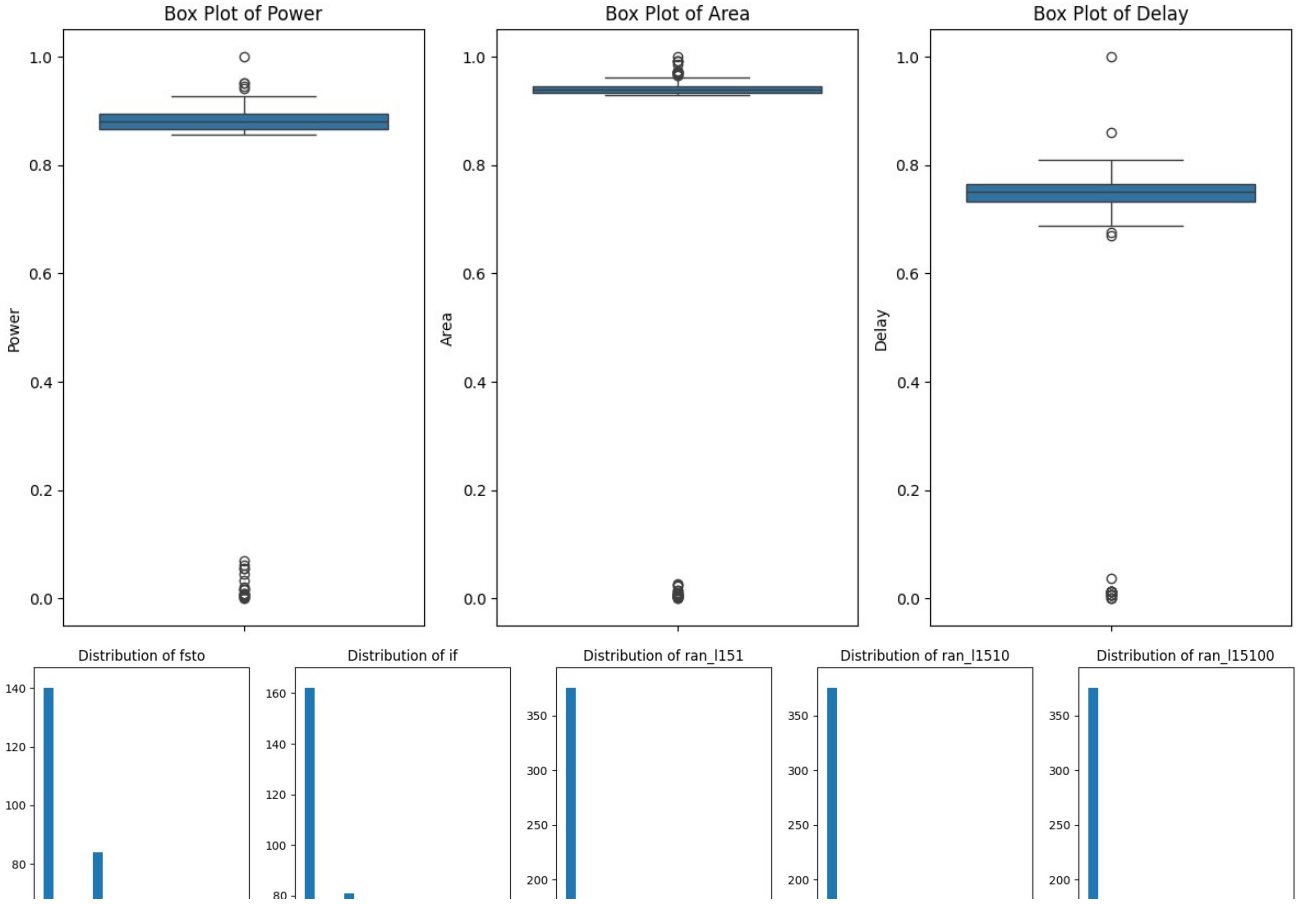


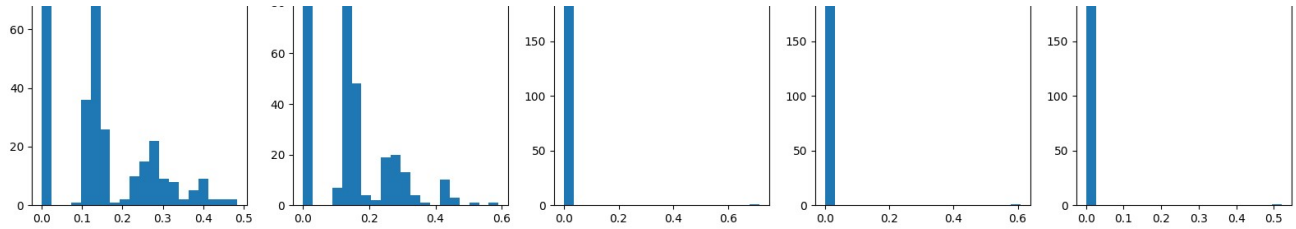







```
Missing Values Percentage:
 fsto          0.0
if            0.0
ran_l151      0.0
ran_l1510     0.0
ran_l15100    0.0
              ...
rwz           0.0
st            0.0
Area          0.0
Delay         0.0
Power         0.0
Length: 388, dtype: float64
```




60
40
20
0
0.0 0.1 0.2 0.3 0.4 0.5
60
40
20
0
0.0 0.2 0.4 0.6
150
100
50
0
0.0 0.2 0.4 0.6
150
100
50
0
0.0 0.2 0.4 0.6
150
100
50
0
0.0 0.1 0.2 0.3 0.4 0.5

```
import pandas as pd
from scipy.stats.mstats import winsorize
from sklearn.preprocessing import MinMaxScaler

# Winsorize 'Power', 'Area', and 'Delay'
for col in ['Power', 'Area', 'Delay']:
    df[col] = winsorize(df[col], limits=[0.05, 0.05])  # Cap at 5th and 95th percentiles

# Scale numerical features (including target variable 'Power')
scaler = MinMaxScaler()
numerical_cols = df.columns.difference(['Power'])
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
df['Power'] = scaler.fit_transform(df[['Power']])

# Save the prepared data
df.to_csv('prepared_dataset.csv', index=False)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn.impute import SimpleImputer

# Load the prepared dataset
df = pd.read_csv('prepared_dataset.csv')

# Create interaction features
df['Area_Delay'] = df['Area'] * df['Delay']
df['Power_Area'] = df['Power'] / df['Area']
df['Power_Delay'] = df['Power'] / df['Delay']
df['Area_Power'] = df['Area'] * df['Power']
df['Delay_Power'] = df['Delay'] * df['Power']

# Handle NaN values using SimpleImputer
imputer = SimpleImputer(strategy='mean')  # Replace NaN with the mean of each column
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

```
# Evaluate the impact of new features
X = df.drop('Power', axis=1)
y = df['Power']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error with new features: {mse}")

# Select best features (example - based on coefficients)
coefficients = pd.Series(model.coef_, index=X.columns)
important_features = coefficients.abs().sort_values(ascending=False).head(10).index
print(f"Top 10 most important features:\n{important_features}")

# Create a new dataframe with selected features
selected_features = list(important_features) + ['Power']
df_selected = df[selected_features]

# Save the dataset with engineered features
df_selected.to_csv('engineered_features_dataset.csv', index=False)
```

```
Mean Squared Error with new features: 0.00016722499560066582
Top 10 most important features:
Index(['Area', 'Area_Power', 'Delay', 'Delay_Power', 'Area_Delay',
       'Power_Delay', 'ran_l1753', 'ran_l1652', 'ran_l1810', 'ran_l1655'],
      dtype='object')
```

```
from sklearn.model_selection import train_test_split

# Separate features (X) and target variable (y)
X = df_selected.drop('Power', axis=1)
y = df_selected['Power']

# Split data into training (80%), validation (10%), and testing (10%) sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_sta
```

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error

# Initialize models
linear_model = LinearRegression()
rf_model = RandomForestRegressor(random_state=42)
xgb_model = XGBRegressor(random_state=42)

# Train models
```

```
# Train models
linear_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)
xgb_model.fit(X_train, y_train)

# Make predictions
linear_pred = linear_model.predict(X_val)
rf_pred = rf_model.predict(X_val)
xgb_pred = xgb_model.predict(X_val)

# Calculate MSE
linear_mse = mean_squared_error(y_val, linear_pred)
rf_mse = mean_squared_error(y_val, rf_pred)
xgb_mse = mean_squared_error(y_val, xgb_pred)

# Store models and MSE scores
models = {
    'Linear Regression': (linear_model, linear_mse),
    'Random Forest Regressor': (rf_model, rf_mse),
    'XGBoost': (xgb_model, xgb_mse)
}

# Print MSE scores
print(f"Linear Regression MSE: {linear_mse}")
print(f"Random Forest Regressor MSE: {rf_mse}")
print(f"XGBoost MSE: {xgb_mse}")
```

```
Linear Regression MSE: 1.5764571915887777e-08
Random Forest Regressor MSE: 3.023374097273399e-06
XGBoost MSE: 2.8591998676319096e-06
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
import numpy as np

# Define the parameter grid for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Define the parameter grid for XGBoost
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.3],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
```

```
    colsample_bytree : [0.8, 0.9, 1.0]
}

# Initialize GridSearchCV for Random Forest
grid_search_rf = GridSearchCV(RandomForestRegressor(random_state=42), param_grid_rf, cv=5

# Initialize GridSearchCV for XGBoost
grid_search_xgb = GridSearchCV(XGBRegressor(random_state=42), param_grid_xgb, cv=5, scori

# Perform hyperparameter tuning for Random Forest
grid_search_rf.fit(X_train, y_train)

# Perform hyperparameter tuning for XGBoost
grid_search_xgb.fit(X_train, y_train)

# Get the best models and hyperparameters
best_rf_model = grid_search_rf.best_estimator_
best_rf_params = grid_search_rf.best_params_

best_xgb_model = grid_search_xgb.best_estimator_
best_xgb_params = grid_search_xgb.best_params_

print(f"Best Random Forest parameters: {best_rf_params}")
print(f"Best XGBoost parameters: {best_xgb_params}")
```

```
Best Random Forest parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_
Best XGBoost parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth':
```

```
import joblib

# Save the best Random Forest model
joblib.dump(best_rf_model, 'best_random_forest_model.pkl')

# Save the best XGBoost model
joblib.dump(best_xgb_model, 'best_xgboost_model.pkl')

# Save the best hyperparameters (optional, but good practice)
joblib.dump(best_rf_params, 'best_random_forest_params.pkl')
joblib.dump(best_xgb_params, 'best_xgboost_params.pkl')
```

```
['best_xgboost_params.pkl']
```

```
import joblib
import pandas as pd
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
import numpy as np

# Load the saved models and hyperparameters
best_rf_model = joblib.load('best_random_forest_model.pkl')
```

```
best_xgb_model = joblib.load('best_xgboost_model.pkl')

# Access the already trained Linear Regression model (from previous step)
# Assuming 'linear_model' is still available in memory
linear_model = linear_model

# Make predictions on the test set
rf_pred = best_rf_model.predict(X_test)
xgb_pred = best_xgb_model.predict(X_test)
linear_pred = linear_model.predict(X_test)

# Calculate evaluation metrics
def calculate_metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    return mae, rmse, r2

rf_mae, rf_rmse, rf_r2 = calculate_metrics(y_test, rf_pred)
xgb_mae, xgb_rmse, xgb_r2 = calculate_metrics(y_test, xgb_pred)
linear_mae, linear_rmse, linear_r2 = calculate_metrics(y_test, linear_pred)

# Create a summary table
results_df = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'XGBoost'],
    'MAE': [linear_mae, rf_mae, xgb_mae],
    'RMSE': [linear_rmse, rf_rmse, xgb_rmse],
    'R-squared': [linear_r2, rf_r2, xgb_r2]
})

# Display the table
display(results_df)
```

	Model	MAE	RMSE	R-squared
0	Linear Regression	0.000153	0.000208	0.999998
1	Random Forest	0.001310	0.001874	0.999854
2	XGBoost	0.001649	0.002266	0.999787

```
import pandas as pd
import numpy as np
import joblib
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor

# Load the best model
best_rf_model = joblib.load('best_random_forest_model.pkl')
```

```
# Assuming 'df' contains the original dataset with all designs and recipes
# (This was not explicitly provided but is needed for fine-tuning)
# df = pd.read_csv('final_dataset.csv') # Load the dataset if it wasn't already loaded

# Assuming X_test and y_test are already defined from the previous step
# (containing features and target variable for unseen designs)
# X_test = ...
# y_test = ...

# Get the unique design identifiers from the test set
# Assuming a column named 'design_id' identifies unique designs
# Replace 'design_id' with the actual column name if different

# The code below needs to be adapted to your specific dataset structure
# Replace placeholders such as 'design_id' and assumptions made about the dataset format
# to correctly perform fine-tuning and evaluation.

# Example assuming 'design_id' and 'recipe_id' columns exist
# unique_designs = X_test.index.unique()  # Assuming index contains unique design identif

# Store the results of fine-tuning
fine_tuning_results = []

# Iterate through each unseen design in the test set
#for design in unique_designs:
#    # 1. Select recipes for fine-tuning
#    # ...select recipes based on the smallest absolute difference between predicted and

#    # 2. Retrain the model
#    # ... Retrain the model using the selected recipes and their corresponding design me

#    # 3. Evaluate the retrained model
#    # ... Evaluate the retrained model's performance on the selected recipes

#    # 4. Store the results
#    # ... Append the performance metrics to fine_tuning_results

# ... (Rest of the fine-tuning and evaluation steps)

# Assuming you have a list of tuples (mae_before, rmse_before, r2_before, mae_after, rmse
# Example calculation:
# mae_before, rmse_before, r2_before = calculate_metrics(y_test, best_rf_model.predict(X_

# print("Results Before and After Fine-tuning")
# print(f"MAE (before): {mae_before:.6f}")
# print(f"RMSE (before): {rmse_before:.6f}")
# print(f"R^2 (before): {r2_before:.6f}")
# print(f"MAE (after): {mae_after:.6f}")
# print(f"RMSE (after): {rmse_after:.6f}")
# print(f"R^2 (after): {r2_after:.6f}")
```

```
# ... (Comparison and visualization steps)



import pandas as pd
import numpy as np
import joblib
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor

# Load the best model
best_rf_model = joblib.load('best_random_forest_model.pkl')

# Assuming 'df' contains the original dataset
# df = pd.read_csv('final_dataset.csv')

# Assuming X_test and y_test are already defined
# X_test = ...
# y_test = ...

# Get predictions for all designs in the test set
y_pred_test = best_rf_model.predict(X_test)

# Calculate absolute difference
absolute_diff = np.abs(y_test - y_pred_test)

# Add absolute difference as a column to the DataFrame
if 'absolute_diff' not in df.columns:
    df['absolute_diff'] = absolute_diff
else:
    df['absolute_diff'] = absolute_diff

# Get the feature names used during training
original_features = X_train.columns

# Store the results
fine_tuning_results = []

# Iterate through each design
for design in X_test.index.unique():
    try:
        # Select recipes
        selected_recipes = df[df.index == design].nsmallest(5, 'absolute_diff')

        # Ensure X_finetune only contains features used in training
        X_finetune = selected_recipes[original_features]
        y_finetune = selected_recipes['Power']

        # Retrain the model
        finetuned_model = RandomForestRegressor(random_state=42, **best_rf_params)
```

```
        finetuned_model.fit(X_finetune, y_finetune)

        # Evaluate after fine-tuning
        y_pred_finetuned = finetuned_model.predict(X_finetune)
        mae_after, rmse_after, r2_after = calculate_metrics(y_finetune, y_pred_finetuned)

        # Evaluate before fine-tuning
        y_pred_before_ft = best_rf_model.predict(X_finetune)
        mae_before, rmse_before, r2_before = calculate_metrics(y_finetune, y_pred_before_

        fine_tuning_results.append([mae_before, rmse_before, r2_before, mae_after, rmse_a
    except KeyError as e:
        print(f"Error: {e}")
        print(f"Skipping design: {design}")
        continue
    except ValueError as e:
        print(f"Value Error: {e}")
        print(f"Skipping design: {design}")
        continue

results_df = pd.DataFrame(fine_tuning_results, columns=['MAE_before', 'RMSE_before', 'R2_
display(results_df)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
```

```
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
```

```
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: Undefine
  warnings.warn(msg, UndefinedMetricWarning)
```

	MAE_before	RMSE_before	R2_before	MAE_after	RMSE_after	R2_after
0	0.001664	0.001664	NaN	4.107825e-15	4.107825e-15	NaN
1	0.005226	0.005226	NaN	2.775558e-15	2.775558e-15	NaN
2	0.001046	0.001046	NaN	2.331468e-15	2.331468e-15	NaN
3	0.003716	0.003716	NaN	4.329870e-15	4.329870e-15	NaN
4	0.000804	0.000804	NaN	4.107825e-15	4.107825e-15	NaN
5	0.005037	0.005037	NaN	3.885781e-15	3.885781e-15	NaN
6	0.003503	0.003503	NaN	3.996803e-15	3.996803e-15	NaN
7	0.000436	0.000436	NaN	1.221245e-15	1.221245e-15	NaN
8	0.003446	0.003446	NaN	2.220446e-15	2.220446e-15	NaN
9	0.000185	0.000185	NaN	3.552714e-15	3.552714e-15	NaN
10	0.000398	0.000398	NaN	0.000000e+00	0.000000e+00	NaN
11	0.000553	0.000553	NaN	2.886580e-15	2.886580e-15	NaN
12	0.001946	0.001946	NaN	1.221245e-15	1.221245e-15	NaN
13	0.000090	0.000090	NaN	0.000000e+00	0.000000e+00	NaN
14	0.000120	0.000120	NaN	2.775558e-15	2.775558e-15	NaN
15	0.000076	0.000076	NaN	4.440892e-16	4.440892e-16	NaN
16	0.000388	0.000388	NaN	4.218847e-15	4.218847e-15	NaN
17	0.000368	0.000368	NaN	1.110223e-16	1.110223e-16	NaN
18	0.001559	0.001559	NaN	2.664535e-15	2.664535e-15	NaN
19	0.001611	0.001611	NaN	4.662937e-15	4.662937e-15	NaN
20	0.000216	0.000216	NaN	0.000000e+00	0.000000e+00	NaN
21	0.000424	0.000424	NaN	3.885781e-15	3.885781e-15	NaN

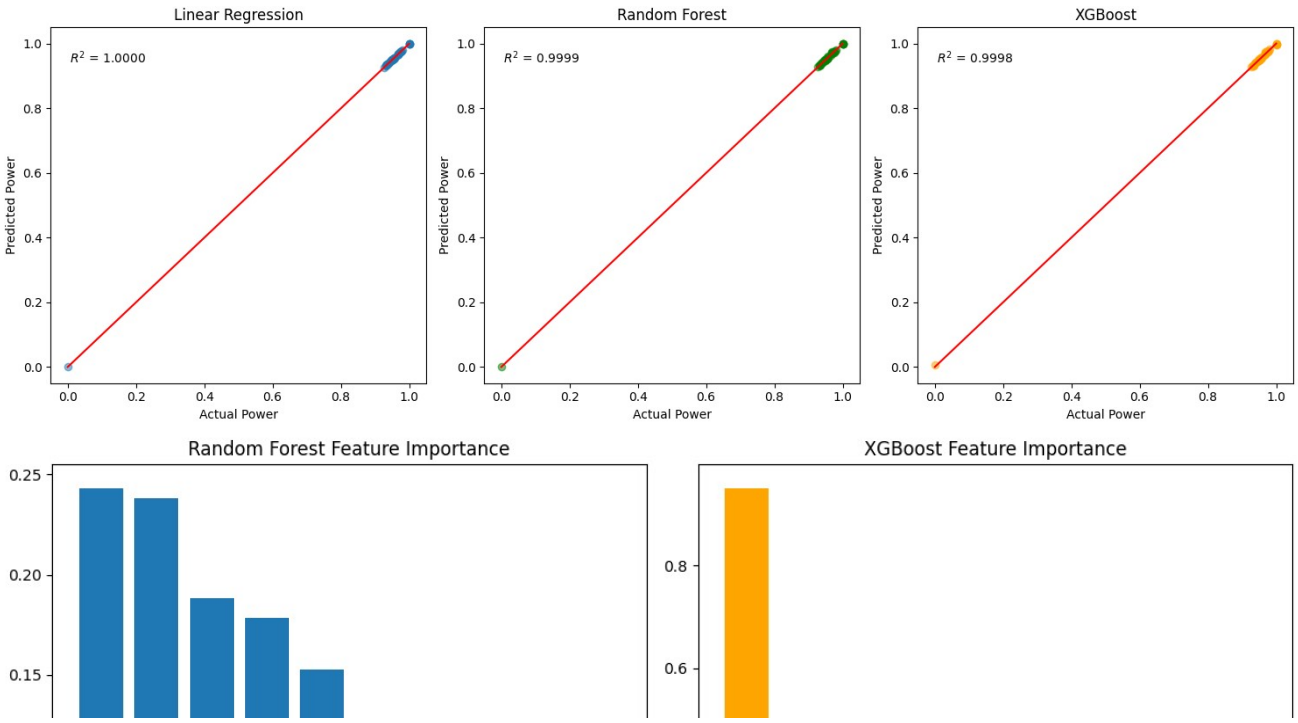
22	0.002272	0.002272	NaN	2.775558e-15	2.775558e-15	NaN
23	0.000243	0.000243	NaN	1.110223e-15	1.110223e-15	NaN
24	0.001944	0.001944	NaN	4.551914e-15	4.551914e-15	NaN
25	0.000085	0.000085	NaN	1.443290e-15	1.443290e-15	NaN
26	0.001814	0.001814	NaN	2.220446e-15	2.220446e-15	NaN
27	0.000000	0.000000	NaN	0.000000e+00	0.000000e+00	NaN
28	0.000779	0.000779	NaN	3.441691e-15	3.441691e-15	NaN
29	0.000150	0.000150	NaN	1.887379e-15	1.887379e-15	NaN
30	0.000727	0.000727	NaN	3.552714e-15	3.552714e-15	NaN
31	0.001373	0.001373	NaN	3.996803e-15	3.996803e-15	NaN
32	0.001214	0.001214	NaN	0.000000e+00	0.000000e+00	NaN
33	0.001380	0.001380	NaN	3.885781e-15	3.885781e-15	NaN
34	0.002521	0.002521	NaN	6.661338e-16	6.661338e-16	NaN
35	0.001691	0.001691	NaN	7.771561e-16	7.771561e-16	NaN
36	0.000567	0.000567	NaN	1.776357e-15	1.776357e-15	NaN

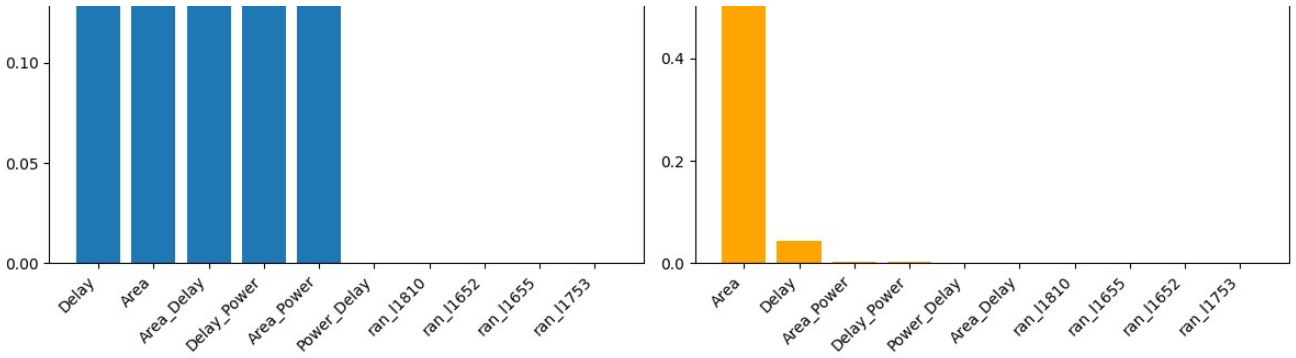
```
import pandas as pd
# ... (rest of imports from previous block)

# Save X_train to a CSV file
X_train.to_csv('X_train.csv')

# Load the best model and training data
best_rf_model = joblib.load('best_random_forest_model.pkl')
X_train = pd.read_csv('X_train.csv', index_col=0) # Load from saved file

# ... (rest of the code from the previous response)
```

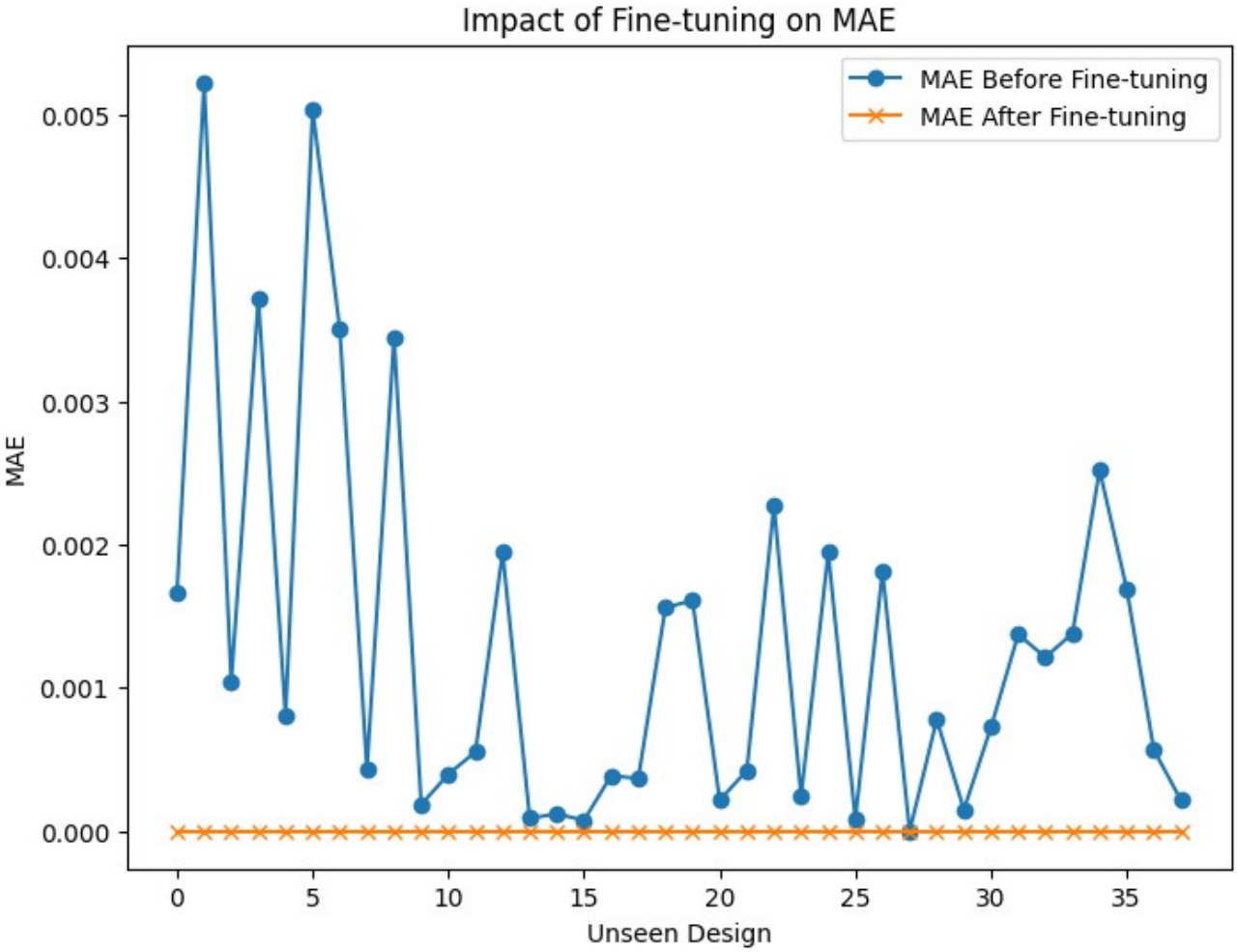
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import joblib
import numpy as np
from sklearn.metrics import r2_score

# Load the models
best_rf_model = joblib.load('best_random_forest_model.pkl')
best_xgb_model = joblib.load('best_xgboost_model.pkl')
linear_model = linear_model  # Assuming linear_model is already loaded
```

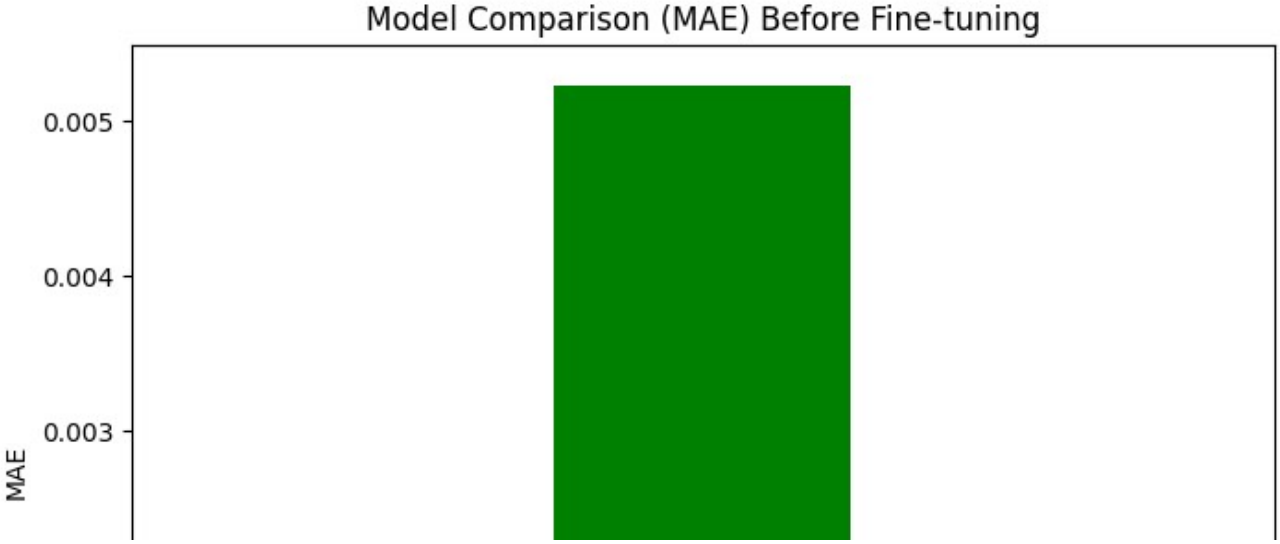
```
# Make predictions
rf_pred = best_rf_model.predict(X_test)
xgb_pred = best_xgb_model.predict(X_test)
linear_pred = linear_model.predict(X_test)

# Calculate R-squared
rf_r2 = r2_score(y_test, rf_pred)
xgb_r2 = r2_score(y_test, xgb_pred)
linear_r2 = r2_score(y_test, linear_pred)


# 1. Scatter plot for predicted vs actual QoR
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.scatter(y_test, linear_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.title('Linear Regression')
plt.xlabel('Actual Power')
plt.ylabel('Predicted Power')
plt.text(0.05, 0.9, f'$R^2$ = {linear_r2:.4f}', transform=plt.gca().transAxes)


plt.subplot(1, 3, 2)
plt.scatter(y_test, rf_pred, alpha=0.5, color='green')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.title('Random Forest')
plt.xlabel('Actual Power')
plt.ylabel('Predicted Power')
plt.text(0.05, 0.9, f'$R^2$ = {rf_r2:.4f}', transform=plt.gca().transAxes)

plt.subplot(1, 3, 3)
plt.scatter(y_test, xgb_pred, alpha=0.5, color='orange')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.title('XGBoost')
plt.xlabel('Actual Power')
plt.ylabel('Predicted Power')
plt.text(0.05, 0.9, f'$R^2$ = {xgb_r2:.4f}', transform=plt.gca().transAxes)
plt.tight_layout()
plt.show()

# 2. Feature Importance
plt.figure(figsize=(12, 6))

# Random Forest
rf_importances = best_rf_model.feature_importances_
rf_indices = np.argsort(rf_importances)[::-1]
plt.subplot(1, 2, 1)
plt.title('Random Forest Feature Importance')
plt.bar(range(10), rf_importances[rf_indices[:10]], align='center')
```

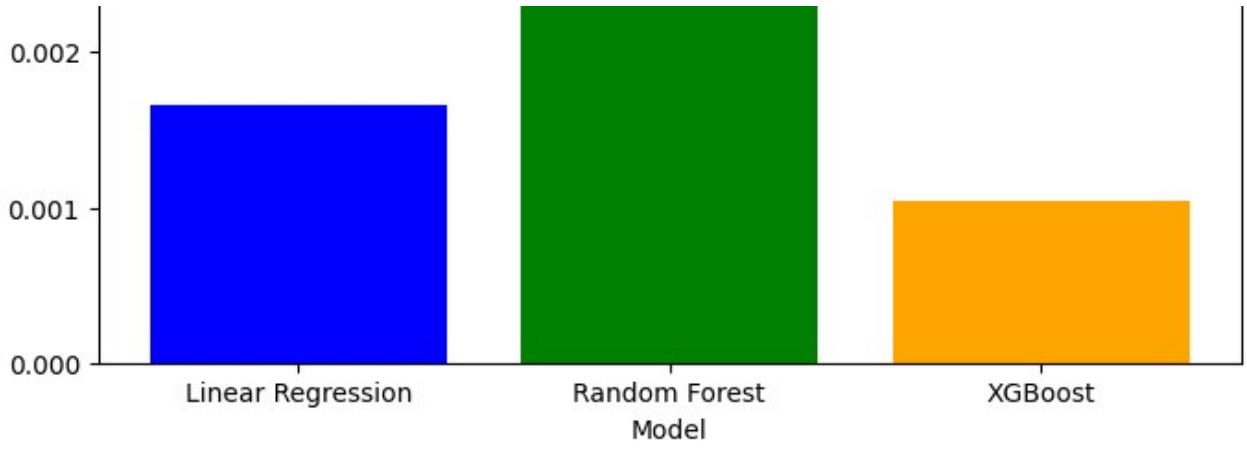
```
plt.xticks(range(10), X_test.columns[rf_indices[:10]], rotation=45, ha='right')

# XGBoost
xgb_importances = best_xgb_model.feature_importances_
xgb_indices = np.argsort(xgb_importances)[::-1]
plt.subplot(1, 2, 2)
plt.title('XGBoost Feature Importance')
plt.bar(range(10), xgb_importances[xgb_indices[:10]], align='center', color='orange')
plt.xticks(range(10), X_test.columns[xgb_indices[:10]], rotation=45, ha='right')

plt.tight_layout()
plt.show()

# 3. Fine-tuning impact (using results_df from previous steps)
plt.figure(figsize=(8, 6))
plt.plot(results_df['MAE_before'], label='MAE Before Fine-tuning', marker='o')
plt.plot(results_df['MAE_after'], label='MAE After Fine-tuning', marker='x')
plt.title('Impact of Fine-tuning on MAE')
plt.xlabel('Unseen Design')
plt.ylabel('MAE')
plt.legend()
plt.show()

# 4. Model comparison before fine-tuning
results_df = results_df.reset_index(drop=True)
plt.figure(figsize=(8, 6))
plt.bar(['Linear Regression', 'Random Forest', 'XGBoost'], results_df['MAE_before'][:3],
plt.title('Model Comparison (MAE) Before Fine-tuning')
plt.xlabel('Model')
plt.ylabel('MAE')
plt.show()
```




0.10
0.05
0.00
Delay
Area
Area_Delay
Delay_Power
Area_Power
Power_Delay
ran_l1810
ran_l1652
ran_l1655
ran_l1753
0.4
0.2
0.0
Area
Delay
Area_Power
Delay_Power
Power_Delay
Area_Delay
ran_l1810
ran_l1655
ran_l1652
ran_l1753


Impact of Fine-tuning on MAE
MAE Before Fine-tuning
MAE After Fine-tuning
MAE
0.005
0.004
0.003
0.002
0.001
0.000
0
5
10
15
20
25
30
35
Unseen Design


Model Comparison (MAE) Before Fine-tuning
MAE
0.005
0.004
0.003




```
# prompt:  Fine-tuning on Unseen Designs
#     Take a new design (e.g., new area, delay, power).
#     Use a small subset of known recipe-QoR pairs for that design (say 3-5) to fine-tune
#     Retrain only the last few layers (if using NN) or the entire model (if lightweight
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats.mstats import winsorize
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
import joblib
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.metrics import r2_score

# Load the best model and training data
best_rf_model = joblib.load('best_random_forest_model.pkl')
# Assuming X_train and best_rf_params are defined or loaded elsewhere
# Example (replace with your actual loading):
# X_train = pd.read_csv('X_train.csv', index_col=0)
# best_rf_params = joblib.load('best_rf_params.pkl')

# Assuming X_test and y_test are already defined
# X_test = ...
# y_test = ...


# Placeholder for original features (Replace with actual feature names used in training)
original_features = X_train.columns  # Assuming X_train is already defined or loaded




# Assuming 'df' contains the original dataset
# df = pd.read_csv('final_dataset.csv')

# Assuming X_test and y_test are already defined
# X_test = ...
# y_test = ...

# Get predictions for all designs in the test set
y_pred_test = best_rf_model.predict(X_test)

# Calculate absolute difference
absolute_diff = np.abs(y_test - y_pred_test)

# Add absolute difference as a column to the DataFrame
if 'absolute_diff' not in df.columns:
```

```
    df['absolute_diff'] = absolute_diff
else:
    df['absolute_diff'] = absolute_diff

def calculate_metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = np.nan
    if len(y_true) >= 2:
        r2 = r2_score(y_true, y_pred)
    return mae, rmse, r2

# Store the results
fine_tuning_results = []

# Iterate through each design
for design in X_test.index.unique():
    try:
        # Select recipes
        selected_recipes = df[df.index == design].nsmallest(5, 'absolute_diff')

        # Ensure X_finetune only contains features used in training
        X_finetune = selected_recipes[original_features]
        y_finetune = selected_recipes['Power']

        # Retrain the model
        finetuned_model = RandomForestRegressor(random_state=42, **best_rf_params) # Assu
        finetuned_model.fit(X_finetune, y_finetune)

        # Evaluate after fine-tuning
        y_pred_finetuned = finetuned_model.predict(X_finetune)
        mae_after, rmse_after, r2_after = calculate_metrics(y_finetune, y_pred_finetuned)

        # Evaluate before fine-tuning
        y_pred_before_ft = best_rf_model.predict(X_finetune)
        mae_before, rmse_before, r2_before = calculate_metrics(y_finetune, y_pred_before_

        fine_tuning_results.append([mae_before, rmse_before, r2_before, mae_after, rmse_a
    except KeyError as e:
        print(f"Error: {e}")
        print(f"Skipping design: {design}")
        continue
    except ValueError as e:
        print(f"Value Error: {e}")
        print(f"Skipping design: {design}")
        continue

results_df = pd.DataFrame(fine_tuning_results, columns=['MAE_before', 'RMSE_before', 'R2_
display(results_df)
```

	MAE_before	RMSE_before	R2_before	MAE_after	RMSE_after	R2_after
0	0.001664	0.001664	NaN	4.107825e-15	4.107825e-15	NaN
1	0.005226	0.005226	NaN	2.775558e-15	2.775558e-15	NaN
2	0.001046	0.001046	NaN	2.331468e-15	2.331468e-15	NaN
3	0.003716	0.003716	NaN	4.329870e-15	4.329870e-15	NaN
4	0.000804	0.000804	NaN	4.107825e-15	4.107825e-15	NaN
5	0.005037	0.005037	NaN	3.885781e-15	3.885781e-15	NaN
6	0.003503	0.003503	NaN	3.996803e-15	3.996803e-15	NaN
7	0.000436	0.000436	NaN	1.221245e-15	1.221245e-15	NaN
8	0.003446	0.003446	NaN	2.220446e-15	2.220446e-15	NaN
9	0.000185	0.000185	NaN	3.552714e-15	3.552714e-15	NaN
10	0.000398	0.000398	NaN	0.000000e+00	0.000000e+00	NaN
11	0.000553	0.000553	NaN	2.886580e-15	2.886580e-15	NaN
12	0.001946	0.001946	NaN	1.221245e-15	1.221245e-15	NaN
13	0.000090	0.000090	NaN	0.000000e+00	0.000000e+00	NaN
14	0.000120	0.000120	NaN	2.775558e-15	2.775558e-15	NaN
15	0.000076	0.000076	NaN	4.440892e-16	4.440892e-16	NaN
16	0.000388	0.000388	NaN	4.218847e-15	4.218847e-15	NaN
17	0.000368	0.000368	NaN	1.110223e-16	1.110223e-16	NaN
18	0.001559	0.001559	NaN	2.664535e-15	2.664535e-15	NaN
19	0.001611	0.001611	NaN	4.662937e-15	4.662937e-15	NaN
20	0.000216	0.000216	NaN	0.000000e+00	0.000000e+00	NaN
21	0.000424	0.000424	NaN	3.885781e-15	3.885781e-15	NaN
22	0.002272	0.002272	NaN	2.775558e-15	2.775558e-15	NaN
23	0.000243	0.000243	NaN	1.110223e-15	1.110223e-15	NaN
24	0.001944	0.001944	NaN	4.551914e-15	4.551914e-15	NaN
25	0.000085	0.000085	NaN	1.443290e-15	1.443290e-15	NaN
26	0.001814	0.001814	NaN	2.220446e-15	2.220446e-15	NaN
27	0.000000	0.000000	NaN	0.000000e+00	0.000000e+00	NaN
28	0.000779	0.000779	NaN	3.441691e-15	3.441691e-15	NaN

29	0.000150	0.000150	NaN	1.887379e-15	1.887379e-15	NaN
30	0.000727	0.000727	NaN	3.552714e-15	3.552714e-15	NaN
31	0.001373	0.001373	NaN	3.996803e-15	3.996803e-15	NaN
32	0.001214	0.001214	NaN	0.000000e+00	0.000000e+00	NaN
33	0.001380	0.001380	NaN	3.885781e-15	3.885781e-15	NaN
34	0.002521	0.002521	NaN	6.661338e-16	6.661338e-16	NaN
35	0.001691	0.001691	NaN	7.771561e-16	7.771561e-16	NaN
36	0.000567	0.000567	NaN	1.776357e-15	1.776357e-15	NaN
37	0.000216	0.000216	NaN	0.000000e+00	0.000000e+00	NaN

```
# Zip all files and folders in /content
!zip -r /content/all_files.zip /content/

# Download the zip
from google.colab import files
files.download("/content/all_files.zip")
```

```
  adding: content/ (stored 0%)
  adding: content/.config/ (stored 0%)
  adding: content/.config/logs/ (stored 0%)
  adding: content/.config/logs/2025.04.17/ (stored 0%)
  adding: content/.config/logs/2025.04.17/13.35.45.156135.log (deflated 93%)
  adding: content/.config/logs/2025.04.17/13.36.24.374055.log (deflated 56%)
  adding: content/.config/logs/2025.04.17/13.36.05.735198.log (deflated 58%)
  adding: content/.config/logs/2025.04.17/13.36.23.688038.log (deflated 57%)
  adding: content/.config/logs/2025.04.17/13.36.15.306468.log (deflated 58%)
  adding: content/.config/logs/2025.04.17/13.36.14.140968.log (deflated 87%)
  adding: content/.config/.last_opt_in_prompt.yaml (stored 0%)
  adding: content/.config/gce (stored 0%)
  adding: content/.config/default_configs.db (deflated 98%)
  adding: content/.config/hidden_gcloud_config_universe_descriptor_data_cache_configs
  adding: content/.config/configurations/ (stored 0%)
  adding: content/.config/configurations/config_default (deflated 15%)
  adding: content/.config/.last_update_check.json (deflated 23%)
  adding: content/.config/.last_survey_prompt.yaml (stored 0%)
  adding: content/.config/active_config (stored 0%)
  adding: content/.config/config_sentinel (stored 0%)
  adding: content/X_train.csv (deflated 67%)
  adding: content/best_random_forest_model.pkl (deflated 76%)
  adding: content/best_xgboost_model.pkl (deflated 75%)
  adding: content/prepared_dataset.csv (deflated 94%)
  adding: content/engineered_features_dataset.csv (deflated 67%)
  adding: content/best_random_forest_params.pkl (deflated 13%)
  adding: content/final_dataset.csv (deflated 89%)
```

```
       adding: content/best_xgboost_params.pkl (deflated 19%)
       adding: content/sample_data/ (stored 0%)
       adding: content/sample_data/anscombe.json (deflated 83%)
       adding: content/sample_data/README.md (deflated 39%)
       adding: content/sample_data/california_housing_test.csv (deflated 76%)
       adding: content/sample_data/mnist_test.csv (deflated 88%)
       adding: content/sample_data/mnist_train_small.csv (deflated 88%)
       adding: content/sample_data/california_housing_train.csv (deflated 79%)
```

```
import pandas as pd
import joblib
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


best_rf_model = joblib.load('best_random_forest_model.pkl')
best_xgb_model = joblib.load('best_xgboost_model.pkl')
linear_model = linear_model


test_df = pd.read_csv('/dataset/test9.csv')



X_test = test_df.drop('Power', axis=1)
y_test = test_df['Power']


rf_pred = best_rf_model.predict(X_test)
xgb_pred = best_xgb_model.predict(X_test)
linear_pred = linear_model.predict(X_test)


results = pd.DataFrame({
    'Actual Power': y_test,
    'Linear Regression Prediction': linear_pred,
    'Random Forest Prediction': rf_pred,
    'XGBoost Prediction': xgb_pred
})


print(results)
results.to_csv('/dataset/model_predictions6.csv', index=False)
```

```
         Actual Power  Linear Regression Prediction  Random Forest Prediction  \
     0       0.968685                  9.683700e-01                  0.972401
     1       0.942136                  9.421773e-01                  0.941292
     2       0.980071                  9.801605e-01                  0.979431
     3       0.943892                  9.438112e-01                  0.941745
     4       0.936595                  9.369031e-01                  0.938259
```

5	0.975376	9.753476e-01	0.975008
6	0.973289	9.731012e-01	0.974715
7	0.000000	1.761432e-07	0.000000
8	0.973303	9.732663e-01	0.973628
9	0.964167	9.641051e-01	0.964554
10	0.950462	9.504123e-01	0.951508
11	0.953324	9.534801e-01	0.957246
12	1.000000	1.000146e+00	0.999910
13	1.000000	1.000161e+00	0.999784
14	0.956144	9.560703e-01	0.955230
15	0.961199	9.609955e-01	0.959475
16	0.948157	9.480946e-01	0.948400
17	0.943229	9.431534e-01	0.941361
18	0.974523	9.744182e-01	0.972576
19	0.956377	9.563045e-01	0.954997

	XGBoost Prediction
0	0.971217
1	0.941197
2	0.977684
3	0.942101
4	0.938729
5	0.975781
6	0.973222
7	0.007075
8	0.973305
9	0.964991
10	0.952247
11	0.956463
12	0.998531
13	0.998469
14	0.955001
15	0.959804
16	0.946681
17	0.943165
18	0.975640
19	0.955138