STEP 1: IMPORT THE NECESSARY LIBRARIES

In [1]:
```
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.linear_model import Ridge, Lasso, ElasticNet, LinearRegressi
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegre
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from xgboost import XGBRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
```

STEP 2: READ THE DATA FROM THE CSV FILES

- [Dataset was downloaded from ourworldindata.org]

In [2]:
```
df1 = pd.read_csv('mental-and-substance-use-as-share-of-disease.csv')
df2 = pd.read_csv('prevalence-by-mental-and-substance-use-disorder.csv')
```

STEP 3: FILL MISSING VALUES IN NUMERIC COLUMNS OF DATAFRAMES df1 AND df2 WITH THE MEAN OF THEIR RESPECTIVE COLUMNS

In [3]:
```
numeric_columns = df1.select_dtypes(include=[np.number]).columns
df1[numeric_columns] = df1[numeric_columns].fillna(df1[numeric_columns].m

numeric_columns = df2.select_dtypes(include=[np.number]).columns
df2[numeric_columns] = df2[numeric_columns].fillna(df2[numeric_columns].m
```

STEP 4: CONVERT DATA TYPES

In [4]:
```
df1['DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Bot
df2['Schizophrenia disorders (share of population) - Sex: Both - Age: Age
df2['Bipolar disorders (share of population) - Sex: Both - Age: Age-stand
df2['Eating disorders (share of population) - Sex: Both - Age: Age-standa
df2['Anxiety disorders (share of population) - Sex: Both - Age: Age-stand
df2['Prevalence - Drug use disorders - Sex: Both - Age: Age-standardized
df2['Depressive disorders (share of population) - Sex: Both - Age: Age-st
df2['Prevalence - Alcohol use disorders - Sex: Both - Age: Age-standardiz
```

STEP 5: MERGE THE TWO DATAFRAMES ON A COMMON COLUMN

In [5]:
```
merged_df = pd.merge(df1, df2, on=['Entity', 'Code', 'Year'])
```

STEP 6: FEATURE THE MATRIX X AND THE VARIABLE y

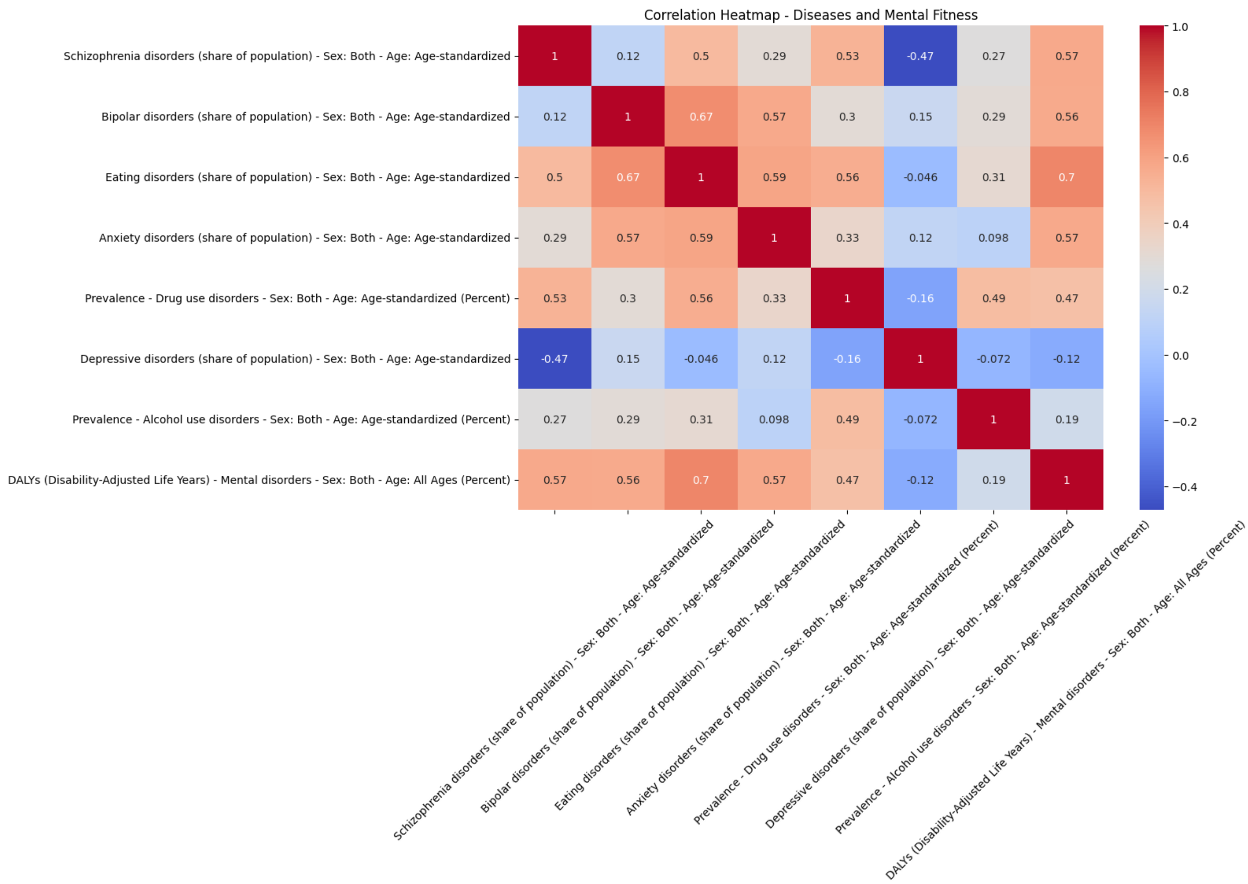
In [6]:
```
X = merged_df[['Schizophrenia disorders (share of population) - Sex: Both
               'Bipolar disorders (share of population) - Sex: Both - Age
               'Eating disorders (share of population) - Sex: Both - Age:
               'Anxiety disorders (share of population) - Sex: Both - Age
               'Prevalence - Drug use disorders - Sex: Both - Age: Age-st
               'Depressive disorders (share of population) - Sex: Both -
               'Prevalence - Alcohol use disorders - Sex: Both - Age: Age

y = merged_df['DALYs (Disability-Adjusted Life Years) - Mental disorders
```

STEP 7: SPLIT THE DATA INTO TRAINING AND TESTING SETS

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

STEP 8: VISUALISING THE CORRELATION HEATMAP OF DISEASES AND MENTAL FITNESS

In [ ]:
```
# Compute the correlation matrix
corr_matrix = merged_df[['Schizophrenia disorders (share of population) -
                          'Bipolar disorders (share of population) - Sex:
                          'Eating disorders (share of population) - Sex: B
                          'Anxiety disorders (share of population) - Sex:
                          'Prevalence - Drug use disorders - Sex: Both - A
                          'Depressive disorders (share of population) - Se
                          'Prevalence - Alcohol use disorders - Sex: Both
                          'DALYs (Disability-Adjusted Life Years) - Mental
                         ]].corr()

# Create the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap - Diseases and Mental Fitness')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```



STEP 9: FIT THE LINEAR REGRESSION MODEL

In [9]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[9]:

```
▾ LinearRegression
LinearRegression()
```

STEP 10: MAKE A PREDICTION USING TRAINED MODEL

In [10]:

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

STEP 11: PRINTING MODEL PERFOMANCE METRICS

In [11]:

```
# Create a dictionary to store the model performance
model_performance = {}

# Ridge Regression
ridge_model = Ridge(alpha=0.5)
ridge_model.fit(X_train, y_train)
ridge_y_pred = ridge_model.predict(X_test)
ridge_mse = mean_squared_error(y_test, ridge_y_pred)
ridge_r2 = r2_score(y_test, ridge_y_pred)
model_performance['1. Ridge Regression'] = {'MSE': ridge_mse, 'R-squared'

# Lasso Regression
lasso_model = Lasso(alpha=0.5)
lasso_model.fit(X_train, y_train)
lasso_y_pred = lasso_model.predict(X_test)
lasso_mse = mean_squared_error(y_test, lasso_y_pred)
lasso_r2 = r2_score(y_test, lasso_y_pred)
model_performance['2. Lasso Regression'] = {'MSE': lasso_mse, 'R-squared'

# Elastic Net Regression
elastic_net_model = ElasticNet(alpha=0.5, l1_ratio=0.5)
elastic_net_model.fit(X_train, y_train)
elastic_net_y_pred = elastic_net_model.predict(X_test)
elastic_net_mse = mean_squared_error(y_test, elastic_net_y_pred)
elastic_net_r2 = r2_score(y_test, elastic_net_y_pred)
model_performance['3. Elastic Net Regression'] = {'MSE': elastic_net_mse,

# Polynomial Regression
poly_features = PolynomialFeatures(degree=2)
X_poly = poly_features.fit_transform(X_train)
poly_model = LinearRegression()
poly_model.fit(X_poly, y_train)
```

```
X_test_poly = poly_features.transform(X_test)
poly_y_pred = poly_model.predict(X_test_poly)
poly_mse = mean_squared_error(y_test, poly_y_pred)
poly_r2 = r2_score(y_test, poly_y_pred)
model_performance['4. Polynomial Regression'] = {'MSE': poly_mse, 'R-squa

# Decision Tree Regression
tree_model = DecisionTreeRegressor()
tree_model.fit(X_train, y_train)
tree_y_pred = tree_model.predict(X_test)
tree_mse = mean_squared_error(y_test, tree_y_pred)
tree_r2 = r2_score(y_test, tree_y_pred)
model_performance['5. Decision Tree Regression'] = {'MSE': tree_mse, 'R-s

# Random Forest Regression
forest_model = RandomForestRegressor()
forest_model.fit(X_train, y_train)
forest_y_pred = forest_model.predict(X_test)
forest_mse = mean_squared_error(y_test, forest_y_pred)
forest_r2 = r2_score(y_test, forest_y_pred)
model_performance['6. Random Forest Regression'] = {'MSE': forest_mse, 'R

# SVR (Support Vector Regression)
svr_model = SVR()
svr_model.fit(X_train, y_train)
svr_y_pred = svr_model.predict(X_test)
svr_mse = mean_squared_error(y_test, svr_y_pred)
svr_r2 = r2_score(y_test, svr_y_pred)
model_performance['7. Support Vector Regression'] = {'MSE': svr_mse, 'R-s

# XGBoost Regression
xgb_model = XGBRegressor()
xgb_model.fit(X_train, y_train)
xgb_y_pred = xgb_model.predict(X_test)
xgb_mse = mean_squared_error(y_test, xgb_y_pred)
xgb_r2 = r2_score(y_test, xgb_y_pred)
model_performance['8. XGBoost Regression'] = {'MSE': xgb_mse, 'R-squared'

# K-Nearest Neighbors Regression
knn_model = KNeighborsRegressor()
knn_model.fit(X_train, y_train)
knn_y_pred = knn_model.predict(X_test)
knn_mse = mean_squared_error(y_test, knn_y_pred)
knn_r2 = r2_score(y_test, knn_y_pred)
model_performance['9. K-Nearest Neighbors Regression'] = {'MSE': knn_mse,

# Bayesian Regression
bayesian_model = BayesianRidge()
bayesian_model.fit(X_train, y_train)
bayesian_y_pred = bayesian_model.predict(X_test)
bayesian_mse = mean_squared_error(y_test, bayesian_y_pred)
bayesian_r2 = r2_score(y_test, bayesian_y_pred)
model_performance['10. Bayesian Regression'] = {'MSE': bayesian_mse, 'R-s
```

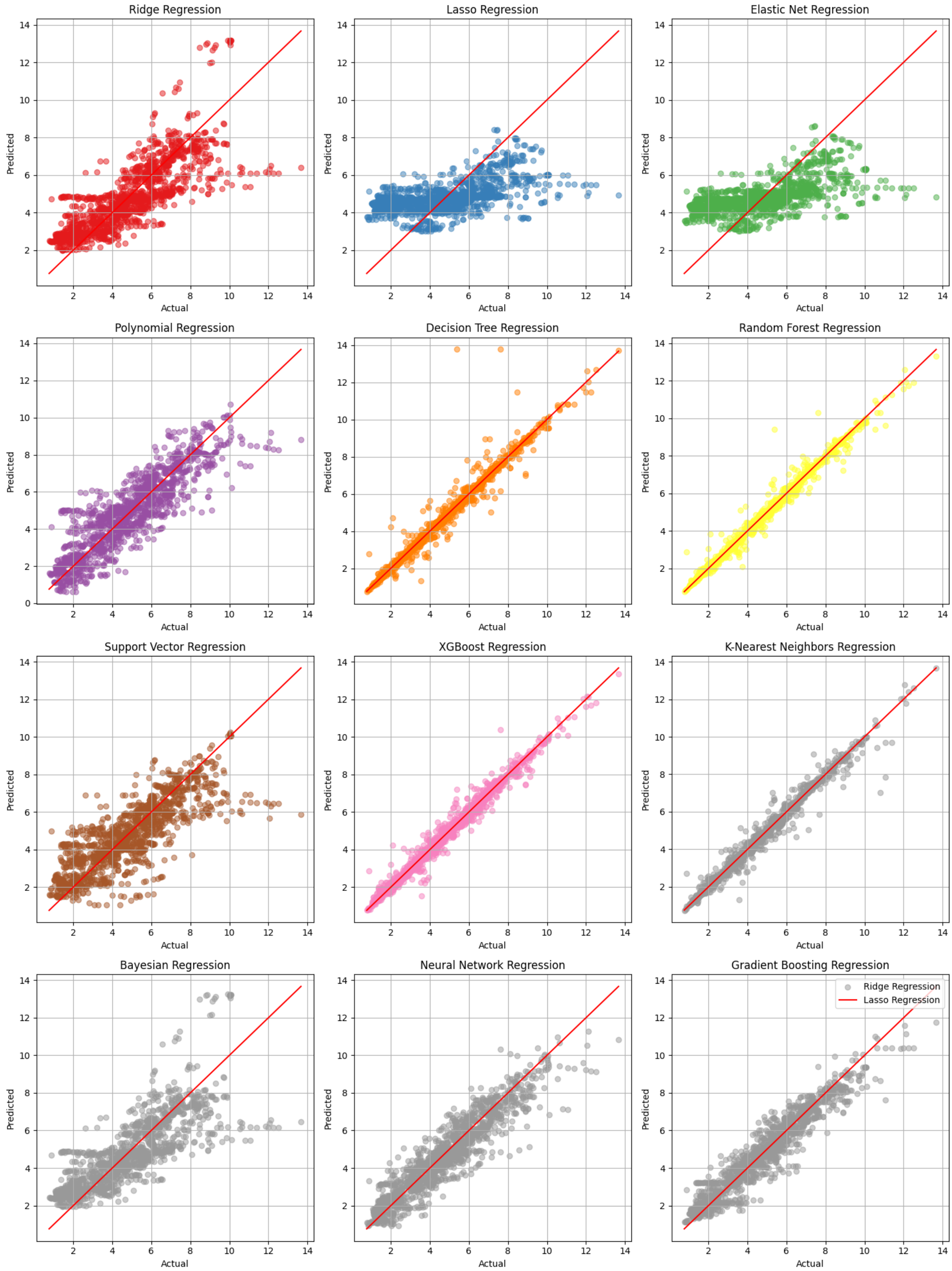
```
# Neural Network Regression
nn_model = MLPRegressor(max_iter=1000)
nn_model.fit(X_train, y_train)
nn_y_pred = nn_model.predict(X_test)
nn_mse = mean_squared_error(y_test, nn_y_pred)
nn_r2 = r2_score(y_test, nn_y_pred)
model_performance['11. Neural Network Regression'] = {'MSE': nn_mse, 'R-s

# Gradient Boosting Regression
gb_model = GradientBoostingRegressor()
gb_model.fit(X_train, y_train)
gb_y_pred = gb_model.predict(X_test)
gb_mse = mean_squared_error(y_test, gb_y_pred)
gb_r2 = r2_score(y_test, gb_y_pred)
model_performance['12. Gradient Boosting Regression'] = {'MSE': gb_mse, '

# Print model performance
for model, performance in model_performance.items():
    print(f"Model: {model}")
    print("   Mean Squared Error (MSE):", performance['MSE'])
    print("   R-squared Score:", performance['R-squared'])
    print()
```

```
Model: 1. Ridge Regression
   Mean Squared Error (MSE): 1.8852828652623428
   R-squared Score: 0.6309285836156879

Model: 2. Lasso Regression
   Mean Squared Error (MSE): 3.674451184301676
   R-squared Score: 0.2806729812205011

Model: 3. Elastic Net Regression
   Mean Squared Error (MSE): 3.4451550539587945
   R-squared Score: 0.325561018531185

Model: 4. Polynomial Regression
   Mean Squared Error (MSE): 1.1568022548912842
   R-squared Score: 0.7735392101864342

Model: 5. Decision Tree Regression
   Mean Squared Error (MSE): 0.18290003163580248
   R-squared Score: 0.9641946707433912

Model: 6. Random Forest Regression
   Mean Squared Error (MSE): 0.07659453630064618
   R-squared Score: 0.9850055105678556

Model: 7. Support Vector Regression
   Mean Squared Error (MSE): 1.7461862488419992
   R-squared Score: 0.6581587601491058

Model: 8. XGBoost Regression
   Mean Squared Error (MSE): 0.10148741123716505
   R-squared Score: 0.9801323698949199

Model: 9. K-Nearest Neighbors Regression
   Mean Squared Error (MSE): 0.10942580994152047
   R-squared Score: 0.9785783134147895

Model: 10. Bayesian Regression
   Mean Squared Error (MSE): 1.8759157254998438
   R-squared Score: 0.6327623368435539

Model: 11. Neural Network Regression
   Mean Squared Error (MSE): 0.6900914019818699
   R-squared Score: 0.8649046167782132

Model: 12. Gradient Boosting Regression
   Mean Squared Error (MSE): 0.45726213047145264
   R-squared Score: 0.9104843176259795
```

STEP 12: PLOTTING PREDECTED vs ACTUAL VALUES GRAPH

In [12]:
```
# Create a dictionary to store the model performance
model_performance = {
    'Ridge Regression': {'Predicted': ridge_y_pred, 'Actual': y_test},
    'Lasso Regression': {'Predicted': lasso_y_pred, 'Actual': y_test},
    'Elastic Net Regression': {'Predicted': elastic_net_y_pred, 'Actual':
    'Polynomial Regression': {'Predicted': poly_y_pred, 'Actual': y_test}
    'Decision Tree Regression': {'Predicted': tree_y_pred, 'Actual': y_te
    'Random Forest Regression': {'Predicted': forest_y_pred, 'Actual': y_
    'Support Vector Regression': {'Predicted': svr_y_pred, 'Actual': y_te
    'XGBoost Regression': {'Predicted': xgb_y_pred, 'Actual': y_test},
    'K-Nearest Neighbors Regression': {'Predicted': knn_y_pred, 'Actual':
    'Bayesian Regression': {'Predicted': bayesian_y_pred, 'Actual': y_tes
    'Neural Network Regression': {'Predicted': nn_y_pred, 'Actual': y_tes
    'Gradient Boosting Regression': {'Predicted': gb_y_pred, 'Actual': y_
}

# Set up figure and axes
num_models = len(model_performance)
num_rows = (num_models // 3) + (1 if num_models % 3 != 0 else 0)
fig, axes = plt.subplots(num_rows, 3, figsize=(15, num_rows * 5))

# Define color palette
color_palette = plt.cm.Set1(range(num_models))

# Iterate over the models and plot the predicted vs actual values
for i, (model, performance) in enumerate(model_performance.items()):
    row = i // 3
    col = i % 3
    ax = axes[row, col] if num_rows > 1 else axes[col]

    # Get the predicted and actual values
    y_pred = performance['Predicted']
    y_actual = performance['Actual']

    # Scatter plot of predicted vs actual values
    ax.scatter(y_actual, y_pred, color=color_palette[i], alpha=0.5, marke

    # Add a diagonal line for reference
    ax.plot([y_actual.min(), y_actual.max()], [y_actual.min(), y_actual.m

    # Set the title and labels
    ax.set_title(model)
    ax.set_xlabel('Actual')
    ax.set_ylabel('Predicted')

    # Add gridlines
```

```
    ax.grid(True)

# Adjust spacing between subplots
fig.tight_layout()

# Create a legend
plt.legend(model_performance.keys(), loc='upper right')

# Show the plot
plt.show()
```


Ridge Regression
Lasso Regression
Elastic Net Regression
Polynomial Regression
Decision Tree Regression
Random Forest Regression
Support Vector Regression
XGBoost Regression
K-Nearest Neighbors Regression
Bayesian Regression
Neural Network Regression
Gradient Boosting Regression
Ridge Regression
Lasso Regression
Predicted
Actual

STEP 13: IT PRINTS REGRESSION MODEL IN ORDER OF PRECISION AND A FINAL RESULT TELLING WHICH REGRESSION MODEL HAS THE MOST PRECISE VALUE AND WHICH REGRESSION MODEL HAS LEAST PRECISE VALUE

In [13]:

```
# Store the regression models and their scores in a dictionary
regression_scores = {
    "Ridge Regression": (ridge_mse, ridge_r2),
    "Elastic Net Regression": (elastic_net_mse, elastic_net_r2),
    "Polynomial Regression": (poly_mse, poly_r2),
    "Random Forest Regression": (forest_mse, forest_r2),
    "Gradient Boosting Regression": (gb_mse, gb_r2),
    "Decision Tree Regression": (tree_mse, tree_r2),
    "Lasso Regression": (lasso_mse, lasso_r2),
    "Support Vector Regression": (svr_mse, svr_r2),
    "XGBoost Regression": (xgb_mse, xgb_r2),
    "K-Nearest Neighbors Regression": (knn_mse, knn_r2),
    "Bayesian Regression": (bayesian_mse, bayesian_r2),
    "Neural Network Regression": (nn_mse, nn_r2),
}

# Sort the regression models based on MSE in ascending order and R-square
sorted_models = sorted(regression_scores.items(), key=lambda x: (x[1][0],

print("Regression Models in Order of Precision:")
for i, (model, scores) in enumerate(sorted_models, start=1):
    print(f"{i}. {model}")
    print("   Mean Squared Error (MSE):", scores[0])
    print("   R-squared Score:", scores[1])
    print()

most_precise_model = sorted_models[0][0]
least_precise_model = sorted_models[-1][0]

print(f"The most precise model is: {most_precise_model}")
print(f"The least precise model is: {least_precise_model}")
```

```
Regression Models in Order of Precision:
1. Random Forest Regression
   Mean Squared Error (MSE): 0.07659453630064618
   R-squared Score: 0.9850055105678556

2. XGBoost Regression
   Mean Squared Error (MSE): 0.10148741123716505
   R-squared Score: 0.9801323698949199

3. K-Nearest Neighbors Regression
   Mean Squared Error (MSE): 0.10942580994152047
   R-squared Score: 0.9785783134147895

4. Decision Tree Regression
   Mean Squared Error (MSE): 0.18290003163580248
   R-squared Score: 0.9641946707433912

5. Gradient Boosting Regression
   Mean Squared Error (MSE): 0.45726213047145264
   R-squared Score: 0.9104843176259795

6. Neural Network Regression
   Mean Squared Error (MSE): 0.6900914019818699
   R-squared Score: 0.8649046167782132

7. Polynomial Regression
   Mean Squared Error (MSE): 1.1568022548912842
   R-squared Score: 0.7735392101864342

8. Support Vector Regression
   Mean Squared Error (MSE): 1.7461862488419992
   R-squared Score: 0.6581587601491058

9. Bayesian Regression
   Mean Squared Error (MSE): 1.8759157254998438
   R-squared Score: 0.6327623368435539

10. Ridge Regression
   Mean Squared Error (MSE): 1.8852828652623428
   R-squared Score: 0.6309285836156879

11. Elastic Net Regression
   Mean Squared Error (MSE): 3.4451550539587945
   R-squared Score: 0.325561018531185

12. Lasso Regression
   Mean Squared Error (MSE): 3.674451184301676
   R-squared Score: 0.2806729812205011

The most precise model is: Random Forest Regression
The least precise model is: Lasso Regression
```