Project Problem Statement

Many individuals struggle with maintaining their mental well-being due to various factors such as stress, anxiety, and depression. However, they often lack effective tools to monitor and track their mental fitness progress over time. This poses a significant challenge in identifying pat- terns, triggers, and effective coping mechanisms to improve their mental health. There is a need for a comprehensive mental fitness tracker that can accurately monitor and assess an individu- al's mental well-being, provide personalized insights and recommendations, and enable users to proactively manage and improve their mental fitness.

Agenda:

To Develop a Mental Fitness Tracker which could help in effectively tracking Mental Health of the people

Provide a mental fitness tracker that can accurately monitor and assess an individual's mental well-being, provide personalized insights and recommendations

Purpose:

- The purpose of the mental fitness tracker is to empower individuals in taking control of their mental well-being. It aims to provide a platform that helps users monitor their mental health, identify patterns and triggers, and discover effective coping mechanisms. By offering personalized insights and recommendations, the tracker encourages usersto proactively manage their mental fitness and improve their overall quality of life.

PROJECT WORK

AI Mental Fitness Tracker: Utilizing Machine Learning for Improved Wellbeing

○ Import Required Libraries

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np #linear algebra
import pandas as pd #data processing, csv file I/o(e.g. pd.read_csv)

# Mount the Google Drive to Google Colab

from google.colab import drive
drive.mount('/content/drive')

import seaborn as sns          # Seaborn is a Python data visualization library based on
matplotlib import matplotlib.pyplot as plt # Matplotlib is a low level graph plotting library
in python that                  serves as a visalization utility
import plotly.express as px    # allows you to create interactive plots with very little code
```

Load and prepare data

```
# prevalence-by-mental-and-substance-use-disorder.csv
df1=pd.read_csv("/content/drive/MyDrive/dataset/mental-and-substance-use-as-share-of-disease.csv")
# mental-and-substance-use-as-share-of-disorder.csv
df2=pd.read_csv("/content/drive/MyDrive/dataset/prevalence-by-mental-and-substance-use-disorder.csv")
# prevalence-by-mental-and-substance-use-disorder.csv
df1.head()
```

	Entity	Code	Year	DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)
0	Afghanistan	AFG	1990	1.696670
1	Afghanistan	AFG	1991	1.734281
2	Afghanistan	AFG	1992	1.791189
3	Afghanistan	AFG	1993	1.776779
4	Afghanistan	AFG	1994	1.712986

# mental-and-substance-use-as-share-of-disorder.csv
df2.head()

	Entity	Code	Year	Prevalence - Schizophrenia - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Bipolar disorder - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Eating disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Anxiety disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Drug use disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Depressive disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Alcohol use disorders - Sex: Both - Age: Age-standardized (Percent)
0	Afghanistan	AFG	1990	0.228979	0.721207	0.131001	4.835127	0.454202	5.125291	0.444036
1	Afghanistan	AFG	1991	0.228120	0.719952	0.126395	4.821765	0.447112	5.116306	0.444250
2	Afghanistan	AFG	1992	0.227328	0.718418	0.121832	4.801434	0.441190	5.106558	0.445501
3	Afghanistan	AFG	1993	0.226468	0.717452	0.117942	4.789363	0.435581	5.100328	0.445958
4	Afghanistan	AFG	1994	0.225567	0.717012	0.114547	4.784923	0.431822	5.099424	0.445779

# Merging two datasets: prevalence-by-mental-and-substance-use-disorder.csv, mental-and- substance-use-as-share-of-disorder.csv
data = pd.merge(df1,df2)
data.head(11)

	Entity	Code	Year	DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)	Prevalence - Schizophrenia - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Bipolar disorder - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Eating disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Anxiety disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Drug use disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Depressive disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Alcohol use disorders - Sex: Both - Age: Age-standardized (Percent)
0	Afghanistan	AFG	1990	1.696670	0.228979	0.721207	0.131001	4.835127	0.454202	5.125291	0.444036
1	Afghanistan	AFG	1991	1.734281	0.228120	0.719952	0.126395	4.821765	0.447112	5.116306	0.444250
2	Afghanistan	AFG	1992	1.791189	0.227328	0.718418	0.121832	4.801434	0.441190	5.106558	0.445501
3	Afghanistan	AFG	1993	1.776779	0.226468	0.717452	0.117942	4.789363	0.435581	5.100328	0.445958
4	Afghanistan	AFG	1994	1.712986	0.225567	0.717012	0.114547	4.784923	0.431822	5.099424	0.445779
5	Afghanistan	AFG	1995	1.738272	0.224713	0.716686	0.111129	4.780851	0.428578	5.098495	0.445422
6	Afghanistan	AFG	1996	1.778098	0.223690	0.716388	0.107786	4.777272	0.426393	5.100580	0.444837
7	Afghanistan	AFG	1997	1.781815	0.222424	0.716143	0.103931	4.775242	0.423720	5.105474	0.443938
8	Afghanistan	AFG	1998	1.729402	0.221129	0.716139	0.100343	4.777377	0.422491	5.113707	0.442665
9	Afghanistan	AFG	1999	1.850988	0.220065	0.716323	0.097946	4.782067	0.421215	5.120480	0.441428
10	Afghanistan	AFG	2000	1.893882	0.219501	0.716534	0.097080	4.785518	0.421224	5.124827	0.440410

Data Cleaning

# Missing values in the dataset
data.isnull().sum()

```
Entity                                                                                   0
Code                                                                                   690
Year                                                                                     0
DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)   0
Prevalence - Schizophrenia - Sex: Both - Age: Age-standardized (Percent)                 0
Prevalence - Bipolar disorder - Sex: Both - Age: Age-standardized (Percent)              0
Prevalence - Eating disorders - Sex: Both - Age: Age-standardized (Percent)              0
Prevalence - Anxiety disorders - Sex: Both - Age: Age-standardized (Percent)             0
Prevalence - Drug use disorders - Sex: Both - Age: Age-standardized (Percent)            0
Prevalence - Depressive disorders - Sex: Both - Age: Age-standardized (Percent)          0
Prevalence - Alcohol use disorders - Sex: Both - Age: Age-standardized (Percent)         0
dtype: int64
```

# Drop the column
data.drop('Code',axis=1,inplace = True)

# View the data
data.head(11)

	Entity	Year	DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)	Prevalence - Schizophrenia - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Bipolar disorder - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Eating disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Anxiety disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Drug use disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Depressive disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Alcohol use disorders - Sex: Both - Age: Age-standardized (Percent)
0	Afghanistan	1990	1.696670	0.228979	0.721207	0.131001	4.835127	0.454202	5.125291	0.444036
1	Afghanistan	1991	1.734281	0.228120	0.719952	0.126395	4.821765	0.447112	5.116306	0.444250
2	Afghanistan	1992	1.791189	0.227328	0.718418	0.121832	4.801434	0.441190	5.106558	0.445501
3	Afghanistan	1993	1.776779	0.226468	0.717452	0.117942	4.789363	0.435581	5.100328	0.445958
4	Afghanistan	1994	1.712986	0.225567	0.717012	0.114547	4.784923	0.431822	5.099424	0.445779
5	Afghanistan	1995	1.738272	0.224713	0.716686	0.111129	4.780851	0.428578	5.098495	0.445422
6	Afghanistan	1996	1.778098	0.223690	0.716388	0.107786	4.777272	0.426393	5.100580	0.444837
7	Afghanistan	1997	1.781815	0.222424	0.716143	0.103931	4.775242	0.423720	5.105474	0.443938
8	Afghanistan	1998	1.729402	0.221129	0.716139	0.100343	4.777377	0.422491	5.113707	0.442665
9	Afghanistan	1999	1.850988	0.220065	0.716323	0.097946	4.782067	0.421215	5.120480	0.441428
10	Afghanistan	2000	1.893882	0.219501	0.716534	0.097080	4.785518	0.421224	5.124827	0.440410

size = row * column, shape = tuple of array dimension(row,column)
data.size,data.shape (68400,(6840,10))

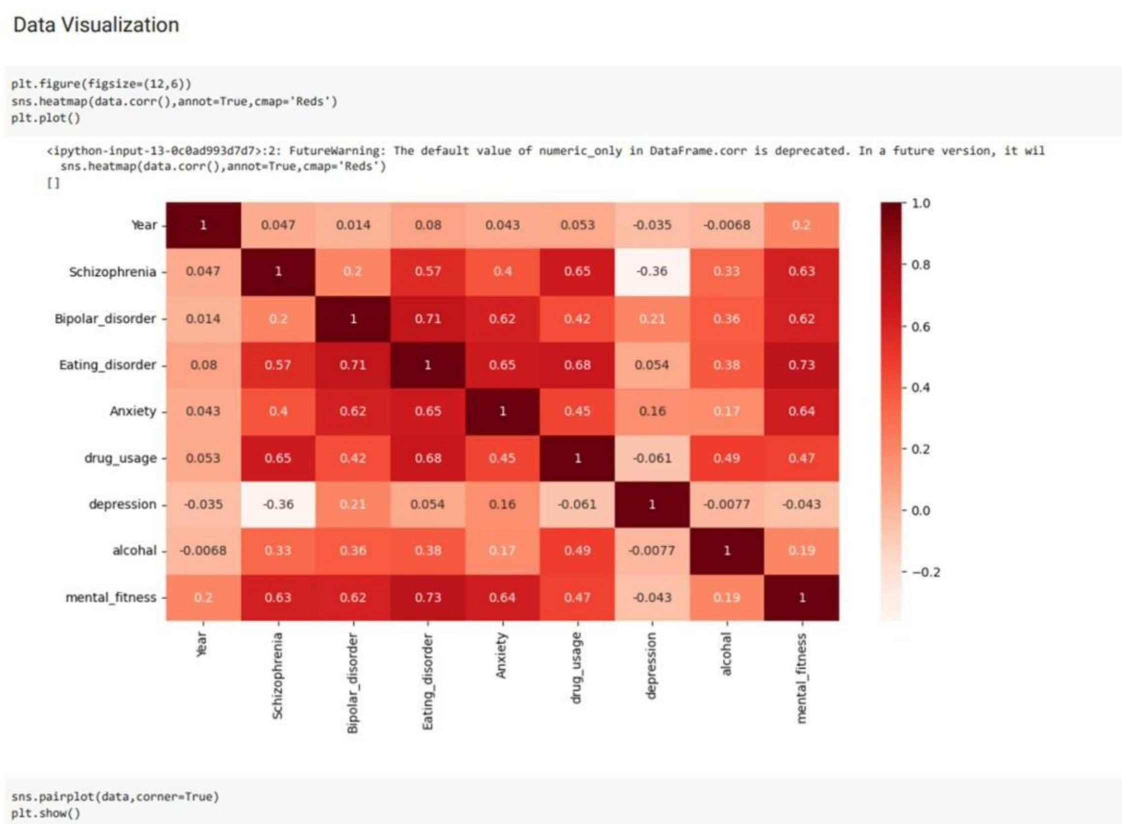
# Column set
data.head(11)

	Entity	Year	DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)	Prevalence - Schizophrenia - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Bipolar disorder - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Eating disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Anxiety disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Drug use disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Depressive disorders - Sex: Both - Age: Age-standardized (Percent)	Prevalence - Alcohol use disorders - Sex: Both - Age: Age-standardized (Percent)
0	Afghanistan	1990	1.696670	0.228979	0.721207	0.131001	4.835127	0.454202	5.125291	0.444036
1	Afghanistan	1991	1.734281	0.228120	0.719952	0.126395	4.821765	0.447112	5.116306	0.444250
2	Afghanistan	1992	1.791189	0.227328	0.718418	0.121832	4.801434	0.441190	5.106558	0.445501
3	Afghanistan	1993	1.776779	0.226468	0.717452	0.117942	4.789363	0.435581	5.100328	0.445958
4	Afghanistan	1994	1.712986	0.225567	0.717012	0.114547	4.784923	0.431822	5.099424	0.445779
5	Afghanistan	1995	1.738272	0.224713	0.716686	0.111129	4.780851	0.428578	5.098495	0.445422
6	Afghanistan	1996	1.778098	0.223690	0.716388	0.107786	4.777272	0.426393	5.100580	0.444837
7	Afghanistan	1997	1.781815	0.222424	0.716143	0.103931	4.775242	0.423720	5.105474	0.443938
8	Afghanistan	1998	1.729402	0.221129	0.716139	0.100343	4.777377	0.422491	5.113707	0.442665
9	Afghanistan	1999	1.850988	0.220065	0.716323	0.097946	4.782067	0.421215	5.120480	0.441428
10	Afghanistan	2000	1.893882	0.219501	0.716534	0.097080	4.785518	0.421224	5.124827	0.440410

□ Visualization

plt.figure(figsize = (12,6))
sns.heatmap(data.corr(),annot=True,cmap='Blues')

# Heatmap is defined as a graphical representation of data using colors to visalize the value of the matrix.
plt.plot()



sns.pairploy(data,corner=True)

# pairwise relationships in a dataset
plt.show()

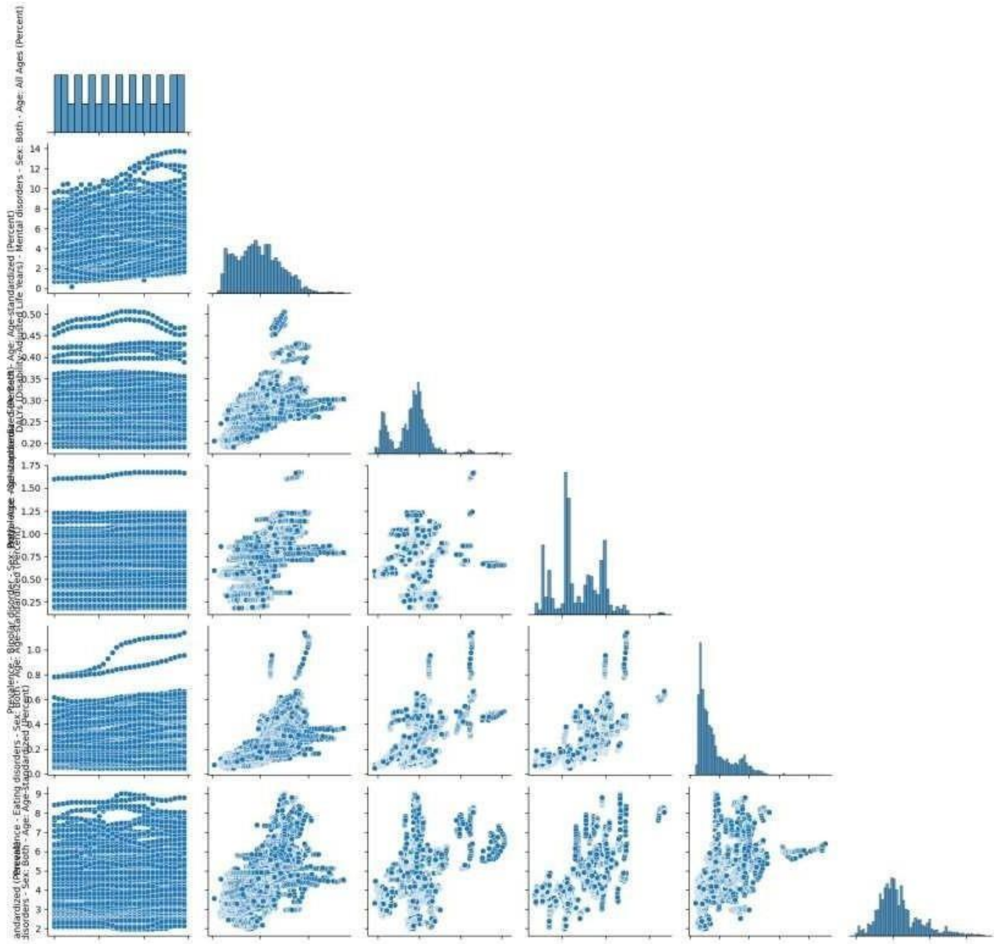
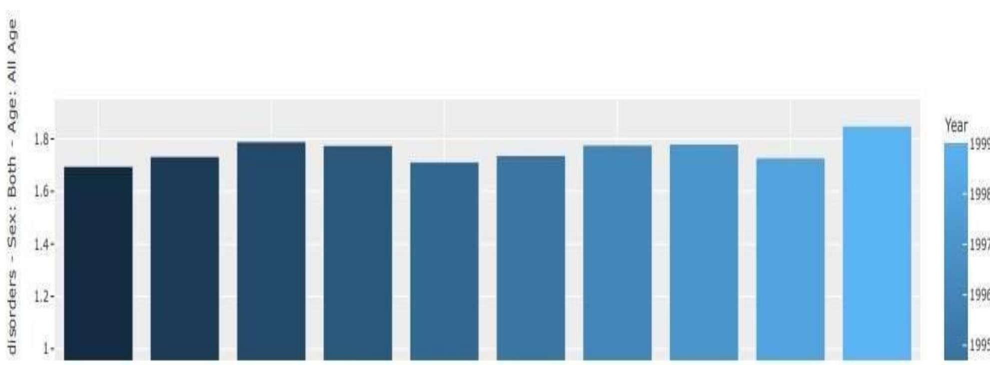


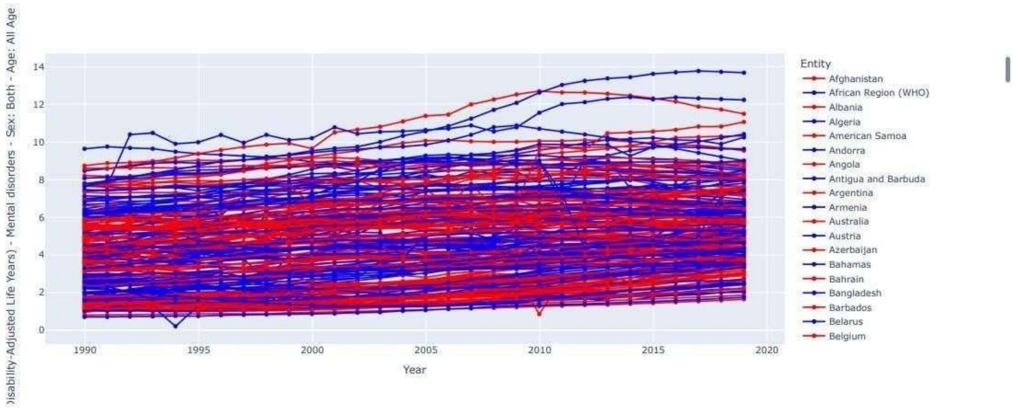
fig = px.bar(data.head(10),x='Year',y='DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)',color='Year',template='ggplot2')
fig.show()



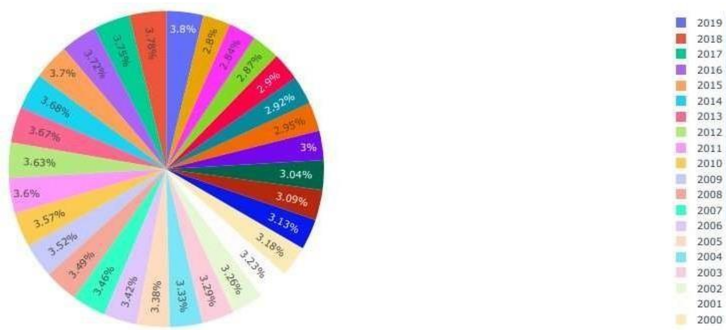
Yearwise variations in mental_fitness of different countries

fig = px.line(data, x='Year' ,y='DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)',

```
markers=True,color='Entity',color_discrete_sequence=["red","blue"])
fig.show()
```




```
fig = px.pie(data, values='DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex:Both - Age: All Ages (Percent)', names='Year')      fig.show()
```



Split Data – (6840,10)

```
x= df.drop('DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age:All Ages (Percent)', axis=1)
x= df['DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)']

from sklearn.model_selection import train_test_split

#use to split the original data into training data & test data
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=20, random_state=2)

#random_state simply sets seed to the random generator, so that your train-test splits are always deterministic. If you don't set seed, it is different each time.
```

```
#  Training  (6840,10)
# 6840*80/100 = 5472
# 6840*20/100 = 1368

print("xstrain: ",xtrain.shape)
print("xtest: ",xtest.shape)
print("\n ystrain: ",ytrain.shape)
print("ytest: ",ytest.shape)

xstrain: (6820, 9)
xtest: (20, 9)

ystrain:(6820,)
ytest:(20,)

# Linear Regression model evaluation for testing set
ytest_pred = lr.predict(xtest)
# using test data(unseen data)
mse=mean_squared_error(ytest,ytest_pred)
rmse =(np.sqrt(mean_squared_error(ytest,ytest_pred)))
r2 = r2_score(ytest,ytest_pred)

print("The Linear Regressor model performance for testing
set")
print("--------------------------------------------------")
print("MSE is {}".format(mse))
print("RMSE is {}".format(rmse))
print("R2 Score is {}".format(r2))

   The Linear Regressor model performance for testing set
   ..................................................................................
   MSE is 0.9579258659002294
   RMSE is  0.9787368726579322
   R2 Score is 0.8132599804568195

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

#  Ridge  Regression
ridge_model = Ridge(alpha=0.5)
ridge_model.fit(xtrain,  ytrain)
ridge_y_pred  =  ridge_model.predict(xtest)
```

```
ridge_mse = mean_squared_error(ytest, ridge_y_pred)
ridge_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
ridge_r2 = r2_score(ytest, ridge_y_pred)
model_performance['1. Ridge Regression'] = {'MSE': ridge_mse, 'RMSE' : ridge_rmse,
'R-squared': ridge_r2}

# Lasso Regression
lasso_model = Lasso(alpha=0.5)
lasso_model.fit(xtrain, ytrain)
lasso_y_pred = lasso_model.predict(xtest)
lasso_mse = mean_squared_error(ytest, lasso_y_pred)
lasso_rmse = (np.sqrt(mean_squared_error(ytest,
ytest_pred)))
lasso_r2 = r2_score(ytest, lasso_y_pred)

model_performance['2. Lasso Regression'] = {'MSE': lasso_mse, 'RMSE': lasso_rmse, 'R-
squared': lasso_r2}

# Elastic Net Regression

elastic_net_model = ElasticNet(alpha=0.5, l1_ratio=0.5)
elastic_net_model.fit(xtrain, ytrain)
elastic_net_y_pred = elastic_net_model.predict(xtest)
elastic_net_mse = mean_squared_error(ytest,
elastic_net_y_pred)
lasso_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
elastic_net_r2 = r2_score(ytest, elastic_net_y_pred)
model_performance['3. Elastic Net Regression'] = {'MSE':
elastic_net_mse, 'RMSE': lasso_rmse,'R-squared':
elastic_net_r2}

# Polynomial Regression     poly_features =
PolynomialFeatures(degree=2)     xpoly =
poly_features.fit_transform(xtrain)
poly_model = LinearRegression()
poly_model.fit(xpoly, ytrain)
X_test_poly = poly_features.transform(xtest)
```

```
poly_y_pred = poly_model.predict(X_test_poly)
poly_mse = mean_squared_error(ytest, poly_y_pred)
poly_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
poly_r2 = r2_score(ytest, poly_y_pred)
model_performance['4. Polynomial Regression'] = {'MSE': poly_mse, 'RMSE': poly_rmse,
'R-squared': poly_r2}

# Decision Tree Regression
tree_model = DecisionTreeRegressor()
tree_model.fit(xtrain, ytrain)
tree_y_pred = tree_model.predict(xtest)
tree_mse = mean_squared_error(ytest, tree_y_pred)
tree_rmse = (np.sqrt(mean_squared_error(ytest,
ytest_pred)))
tree_r2 = r2_score(ytest, tree_y_pred)
model_performance['5. Decision Tree Regression'] = {'MSE': tree_mse, 'RMSE': tree_rmse,
'R-squared': tree_r2}

# Random Forest Regression
forest_model = RandomForestRegressor()
forest_model.fit(xtrain, ytrain)
forest_y_pred = forest_model.predict(xtest)
forest_mse = mean_squared_error(ytest, forest_y_pred)
forest_rmse = (np.sqrt(mean_squared_error(ytest,
ytest_pred)))
forest_r2 = r2_score(ytest, forest_y_pred)
model_performance['6. Random Forest Regression']
={'MSE': forest_mse, 'RMSE': forest_rmse, 'R-squared':
forest_r2}

# SVR (Support Vector Regression)
svr_model = SVR()
svr_model.fit(xtrain, ytrain)
svr_y_pred = svr_model.predict(xtest)
svr_mse = mean_squared_error(ytest, svr_y_pred)
svr_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
svr_r2 = r2_score(ytest, svr_y_pred)
```

```
model_performance['7. Support Vector Regression'] = {'MSE': svr_mse, 'RMSE':
svr_rmse, 'R-squared': svr_r2}

# XGBoost Regression
xgb_model = XGBRegressor()
xgb_model.fit(xtrain, ytrain)
xgb_y_pred = xgb_model.predict(xtest)
xgb_mse =  mean_squared_error(ytest,  xgb_y_pred)
xgb_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
xgb_r2 = r2_score(ytest, xgb_y_pred)model_performance['8. XGBoost Regression'] =
{'MSE': xgb_mse, 'RMSE': xgb_rmse, 'R-     squared': xgb_r2}

# K-Nearest Neighbors Regression
knn_model = KNeighborsRegressor()
knn_model.fit(xtrain, ytrain)
knn_y_pred = knn_model.predict(xtest)
knn_mse =  mean_squared_error(ytest,  knn_y_pred)
knn_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
knn_r2 = r2_score(ytest, knn_y_pred) model_performance['9. K-Nearest Neighbors
Regression'] = {'MSE': knn_mse, 'RMSE': knn_rmse, 'R-squared': knn_r2}


# Bayesian Regression
bayesian_model = BayesianRidge()
bayesian_model.fit(xtrain, ytrain)
bayesian_y_pred =
bayesian_model.predict(xtest)
bayesian_mse = mean_squared_error(ytest, bayesian_y_pred)
bayesion_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
bayesian_r2 = r2_score(ytest, bayesian_y_pred)
model_performance['10. Bayesian Regression'] = {'MSE': bayesian_mse,
'RMSE': bayesion_rmse, 'R-squared': bayesian_r2}

# Neural Network Regression
nn_model = MLPRegressor(max_iter=1000)
nn_model.fit(xtrain, ytrain)
nn_y_pred = nn_model.predict(xtest)
```

```
nn_mse = mean_squared_error(ytest, nn_y_pred)
nn_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
nn_r2 = r2_score(ytest, nn_y_pred) model_performance['11. Neural Network Regression'] =
{'MSE': nn_mse, 'RMSE': nn_rmse,'R-squared': nn_r2}

# Gradient Boosting Regression
gb_model = GradientBoostingRegressor()
gb_model.fit(xtrain, ytrain)
gb_y_pred = gb_model.predict(xtest)
gb_mse = mean_squared_error(ytest, gb_y_pred)
gb_rmse = (np.sqrt(mean_squared_error(ytest, ytest_pred)))
gb_r2 = r2_score(ytest, gb_y_pred) model_performance['12. Gradient Boosting
Regression'] = {'MSE': gb_mse, 'RMSE': gb_rmse, 'R-squared': gb_r2}

# Print model performance for model, performance
in model_performance.items():
print(f"The {model} performance for testing set")
print("--------------------------------------------")
print(" MSE is {}".format(performance['MSE']))
print(" RMSE is {}".format(performance['RMSE']))
print(" R2 Score is {}".format(performance['R-squared']))
print()
```

```
The 1. Ridge Regression performance for testing set
--------------------------------------------
MSE is 1.0098044182897008
RMSE is 0.9787368726579322
R2 Score is 0.8031466697801343

The 2. Lasso Regression performance for testing set
--------------------------------------------
MSE is 2.892667412572883
RMSE is 0.9787368726579322
R2 Score is 0.43609752238171384

The 3. Elastic Net Regression performance for testing set
--------------------------------------------
MSE is 2.8617687447329017
RMSE is 0.9787368726579322
R2 Score is 0.44212097162940056
```

The 4. Polynomial Regression performance for testing set

....

MSE is 0.38171471932240103

RMSE is 0.9787368726579322
R2 Score is 0.9255877550824909

The 5. Decision Tree Regression performance for testing set

....

MSE is 0.03180707066916211

RMSE is 0.9787368726579322
R2 Score is 0.9937994648544244

The 6. Random Forest Regression performance for testing set

....

MSE is 0.009256026616497316

RMSE is 0.9787368726579322
R2 Score is 0.9981956113173408

The 7. Support Vector Regression performance for testing set

....

MSE is 5.195640340336887
RMSE is 0.9787368726579322
R2 Score is -0.012848711191289164

The 8. XGBoost Regression performance for testing set

....

MSE is 0.031606881153924925
RMSE is 0.9787368726579322
R2 Score is 0.9938384902062999

The 9. K-Nearest Neighbors Regression performance for testing set

....

MSE is 0.21874228571103854

RMSE is 0.9787368726579322
R2 Score is 0.9573579332569628

The 10. Bayesian Regression performance for testing set

....

MSE is 0.9596974366457351

RMSE is 0.9787368726579322
R2 Score is 0.8129146268470943

The 11. Neural Network Regression performance for testing set

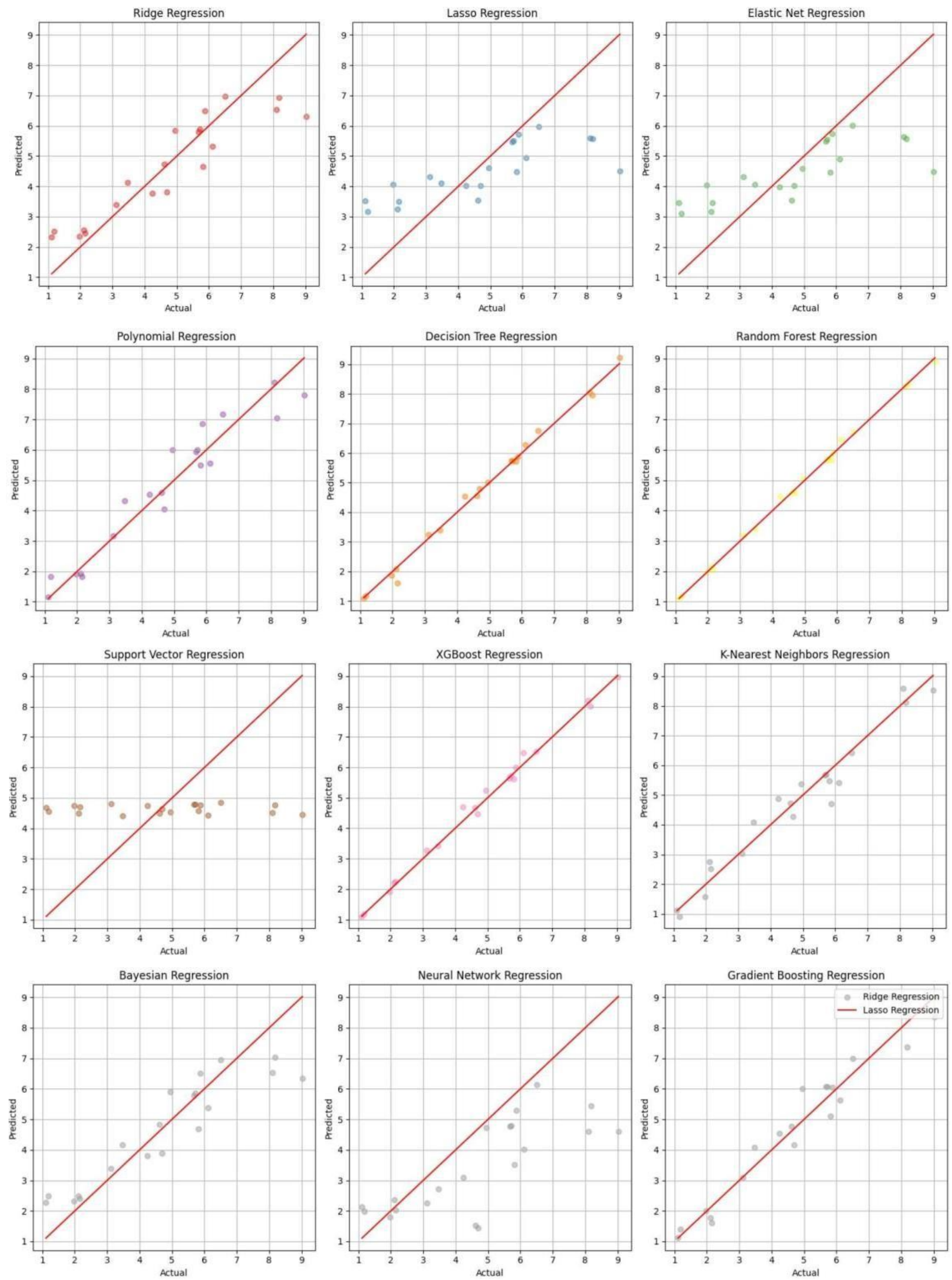
```
MSE is 3.7949312862258857
RMSE is 0.9787368726579322
R2 Score is 0.26020836498775746
```

The 12. Gradient Boosting Regression performance for testing set

```
MSE is 0.24656855340602474
RMSE is 0.9787368726579322
R2 Score is 0.9519334239518589
```

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

# 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, marker='o')


# Add a diagonal line for reference        ax.plot([y_actual.min(), y_actual.max()],
[y_actual.min(), y_actual.max()], color='r')


# 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
Actual
Predicted

Sort the regression models based on MSE in ascending order and R-squared score in descending order

```
sorted_models = sorted(regression_scores.items(), key=lambda x: (x[1][0], -x[1][1]))

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): 0.009256026616497316
   R-squared Score: 0.9981956113173408

2. XGBoost Regression

   Mean Squared Error (MSE): 0.031606881153924925
   R-squared Score: 0.9938384902062999

3. Decision Tree Regression

   Mean Squared Error (MSE): 0.03180707066916211
   R-squared Score: 0.993799464854424

4. K-Nearest Neighbors Regression

   Mean Squared Error (MSE): 0.21874228571103854
   R-squared Score: 0.9573579332569628

5. Gradient Boosting Regression

   Mean Squared Error (MSE): 0.24656855340602474
   R-squared Score: 0.9519334239518589

6. Polynomial Regression

   Mean Squared Error (MSE): 0.38171471932240103
   R-squared Score: 0.9255877550824909

7. Bayesian Regression

   Mean Squared Error (MSE): 0.9596974366457351
   R-squared Score: 0.8129146268470943

8. Ridge Regression

   Mean Squared Error (MSE): 1.0098044182897008
   R-squared Score: 0.8031466697801343

9. Elastic Net Regression

   Mean Squared Error (MSE): 2.8617687447329017
   R-squared Score: 0.44212097162940056

10. Lasso Regression

   Mean Squared Error (MSE): 2.892667412572883
   R-squared Score: 0.43609752238171384

11. Neural Network Regression

   Mean Squared Error (MSE): 3.7949312862258857
   R-squared Score: 0.26020836498775746

12. Support Vector Regression

   Mean Squared Error (MSE): 5.195640340336887
   R-squared Score: -0.012848711191289164

The most precise model is: Random Forest Regression
The least precise model is: Support Vector Regression

Result:

Root mean square error

```
rmse = (np.sqrt(mean_squared_error(ytrain,ytrain_pred)))
```

R2 (goodness of fit of a model)

```
r2 = r2_score(ytrain,ytrain_pred)
```

```
print("The RandomForestRegressor model performance for training set")
print("..........................")
print("MSE is {}".format(mse))
print("RMSE is {}".format(rmse))
print("R2 score is {}".format(r2))
```

```
The RandomForestRegressor model performance for training set
..........................
MSE is 0.00534293770842881
RMSE is 0.07309540141779652
R2 score is 0.9990056639866395
```

REFERENCES

Project References:

https://colab.research.google.com/drive/1ENAuTwpTEJND0BilV4o9Lw6fF-_iQvwQ?usp=sharing