MAY 13 ,2024

regression project

let’s dive into it :)

PRESENTED TO

DR: Amr Amin

ENG:shomo3

PRESENTED BY

fcds team

MAY 13 ,2024

regression project

let’s dive into it :)

1 Youssef Usama
22010299

2 Moataz Elshimy
22011453

3 Abdelhamed Hassan
22010125

4 Mohamed Ahmed Khalaf Yehia
22010387

5 Nahed Refaay Kamal
22010280

6 Asmaa Hamdy Mahmoud
22010051

7 Yassin Salah
22010410

8 Youssef Abdelwahab
22010308

9 Nada Emad Mahmoud
22010283

10 Abdelrahman Moataz
22010149

11 Suhayb Tharwat
22010123

FIRSTLY

ABOUT DATASET

The World Happiness Report is a landmark survey of the state of global happiness. The first report was published in 2012, the second in 2013, the third in 2015, and the fourth in the 2016 Update. The World Happiness 2017, which ranks 155 countries by their happiness levels, was released at the United Nations at an event celebrating International Day of Happiness on March 20th. The report continues to gain global recognition as governments, organizations and civil society increasingly use happiness indicators to inform their policy-making decisions. Leading experts across fields – economics, psychology, survey analysis, national statistics, health, public policy and more – describe how measurements of well-being can be used effectively to assess the progress of nations. The reports review the state of happiness in the world today and show how the new science of happiness explains personal and national variations in happiness.

DATASET

https://www.kaggle.com/datasets/unsdsn/world-happiness

FIRSTLY

INTEGRATE DATA FROM DIFFERENT CSV FILES

- we have 4 datasets each dataset represents the happiness score in a certain year
- at first we chose the columns that we need from each dataset , renamed the columns and combine three of them in 1 dataset and we will use the 4th dataset later for prediction

```python
happiness_2015 = pd.read_csv("../data/2015.csv")
happiness_2015.columns = ['Country', 'Region', 'Happiness_Rank', 'Happiness_Score',
       'Standard Error', 'Economy', 'Family',
       'Health', 'Freedom', 'Trust',
       'Generosity', 'Dystopia_Residual']
columns_2015 = ['Region', 'Standard Error']
new_dropped_2015 = happiness_2015.drop(columns_2015, axis=1)
```

Python

```python
happiness_2016 =  pd.read_csv("../data/2016.csv")
columns_2016 = ['Region', 'Lower Confidence Interval','Upper Confidence Interval' ]
dropped_2016 = happiness_2016.drop(columns_2016, axis=1)
dropped_2016.columns = ['Country', 'Happiness_Rank', 'Happiness_Score','Economy', 'Family',
       'Health', 'Freedom', 'Trust',
       'Generosity', 'Dystopia_Residual']
```

Python

```python
happiness_2017 =  pd.read_csv("../data/2017.csv")
columns_2017 = ['Whisker.high','Whisker.low' ]
dropped_2017 = happiness_2017.drop(columns_2017, axis=1)
dropped_2017.columns = ['Country', 'Happiness_Rank', 'Happiness_Score','Economy', 'Family',
       'Health', 'Freedom', 'Trust',
       'Generosity', 'Dystopia_Residual']
```

Python

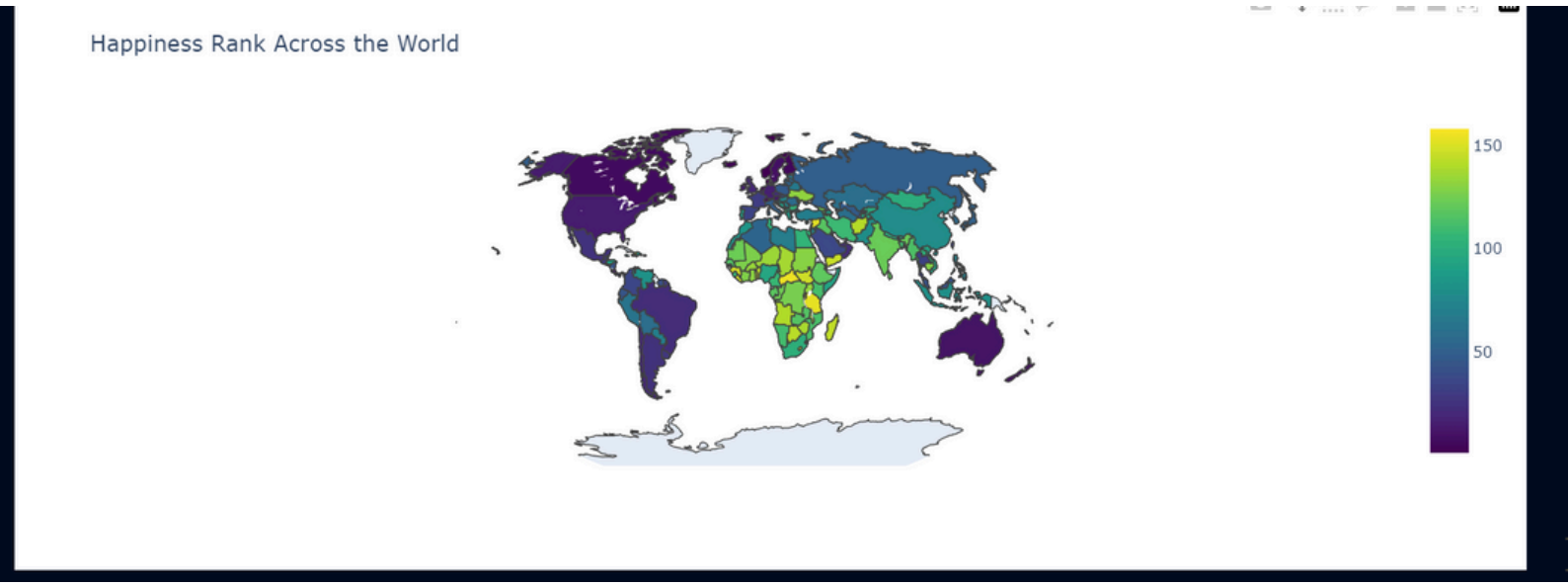
```python
frames = [new_dropped_2015, dropped_2016, dropped_2017]
happiness = pd.concat(frames)
happiness
```

visualisation

heatmap for happines rank

represents the happiness rank for each country when you hover with your mouse

```
earth = dict(type = 'choropleth',
           locations = happiness['Country'],
           locationmode = 'country names',
           z = happiness['Happiness_Rank'],
           text = happiness['Country'],
          colorscale = 'Viridis', reversescale = False)
layout = dict(title = 'Happiness Rank Across the World',
             geo = dict(showframe = False,
                        projection = {'type': 'kavrayskiy7'}))
choromap6 = go.Figure(data = [earth], layout=layout)
iplot(choromap6)
```



checking null values

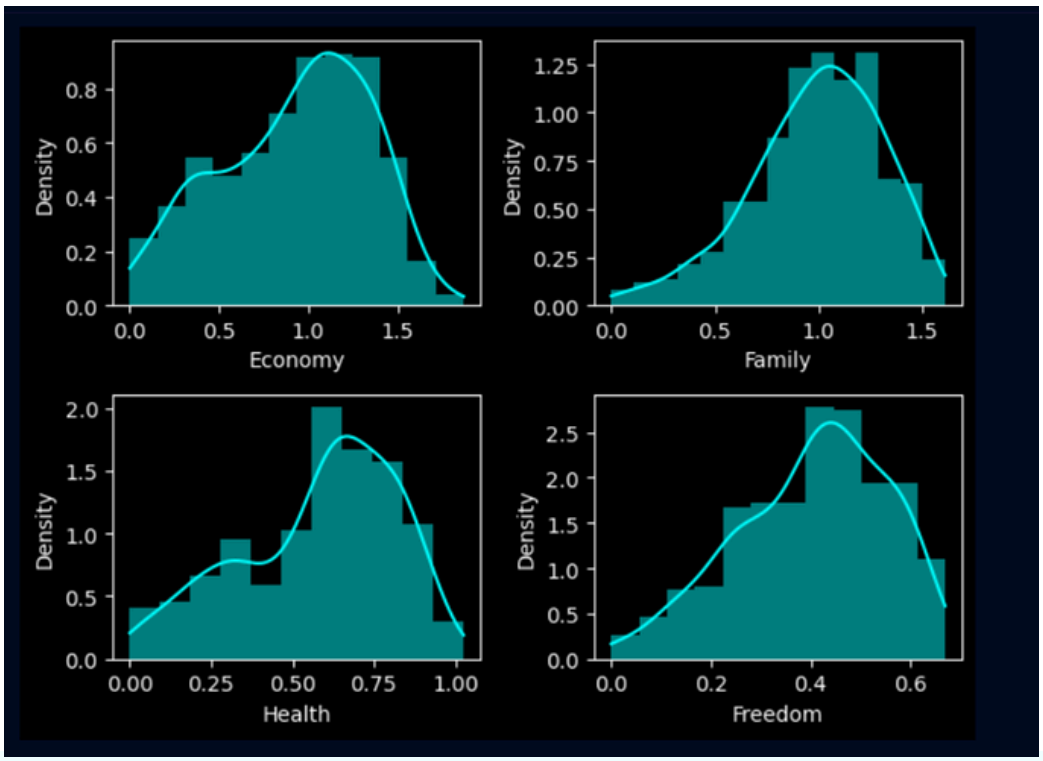
```
Country                0
Happiness_Rank         0
Happiness_Score        0
Economy                0
Family                 0
Health                 0
Freedom                0
Trust                  0
Generosity             0
Dystopia_Residual      0
dtype: int64
```

view our dataset structure

```
happiness.info()
  <class 'pandas.core.frame.DataFrame'>
Index: 470 entries, 0 to 154
Data columns (total 10 columns):
  #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Country             470 non-null    object
 1   Happiness_Rank      470 non-null    int64
 2   Happiness_Score     470 non-null    float64
 3   Economy             470 non-null    float64
 4   Family              470 non-null    float64
 5   Health              470 non-null    float64
 6   Freedom             470 non-null    float64
 7   Trust               470 non-null    float64
 8   Generosity          470 non-null    float64
 9   Dystopia_Residual   470 non-null    float64
dtypes: float64(8), int64(1), object(1)
memory usage: 40.4+ KB
```

ploting featuers dist.

```
plt.subplot(2,2,1)
sns.histplot(happiness['Economy'], color="cyan", kde=True, stat="density",
linewidth=0)
plt.subplot(2,2,2)
sns.histplot(happiness['Family'], color="cyan", kde=True, stat="density",
linewidth=0)
plt.subplot(2,2,3)
sns.histplot(happiness['Health'], color="cyan", kde=True, stat="density",
linewidth=0)
plt.subplot(2,2,4)
sns.histplot(happiness['Freedom'], color="cyan", kde=True, stat="density",
linewidth=0)

plt.tight_layout()
plt.show()


fig = make_subplots(rows=2, cols=2)

fig.add_trace(go.Histogram(x=happiness['Economy'], name='Economy'), row=1,
col=1)
fig.add_trace(go.Histogram(x=happiness['Family'], name='Family'), row=1,
col=2)
fig.add_trace(go.Histogram(x=happiness['Health'], name='Health'), row=2,
col=1)
fig.add_trace(go.Histogram(x=happiness['Freedom'], name='Freedom'), row=2,
col=2)



fig.update_layout(title='Histograms of Economy, Family, Health, and Freedom',
                  barmode='overlay',
                  width=1000, height=800,
                  xaxis=dict(title='Values'),
                  yaxis=dict(title='Density'),template='plotly_dark')

fig.show()
```



ploting featuers dist.

- these are some of the dist. for some features in our dataset we observed that the features are normally distributed
- we observed that we have some outliers in trust and Generosity

outliers analysis

Your paragraph textsubset_df = happiness[['Economy','Health','Family', 'Trust', 'Generosity']]

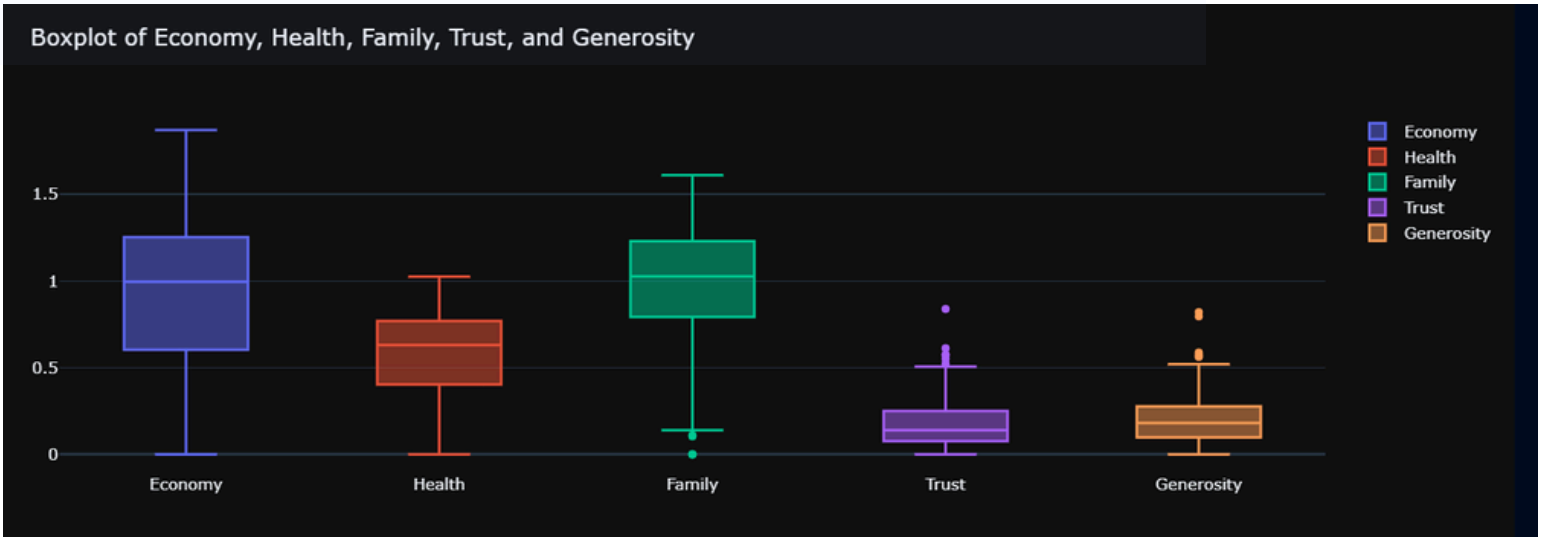
traces = []

for column in subset_df.columns:
trace = go.Box(y=subset_df[column], name=column)
traces.append(trace)

layout = go.Layout(title='Boxplot of Economy, Health, Family, Trust, and Generosity',
yaxis=*dict*(title='Values'))

fig = go.Figure(data=traces, layout=layout)
fig.update_layout(template='plotly_dark')

fig.show()



we observed that we have some outliers in trust and Generosity

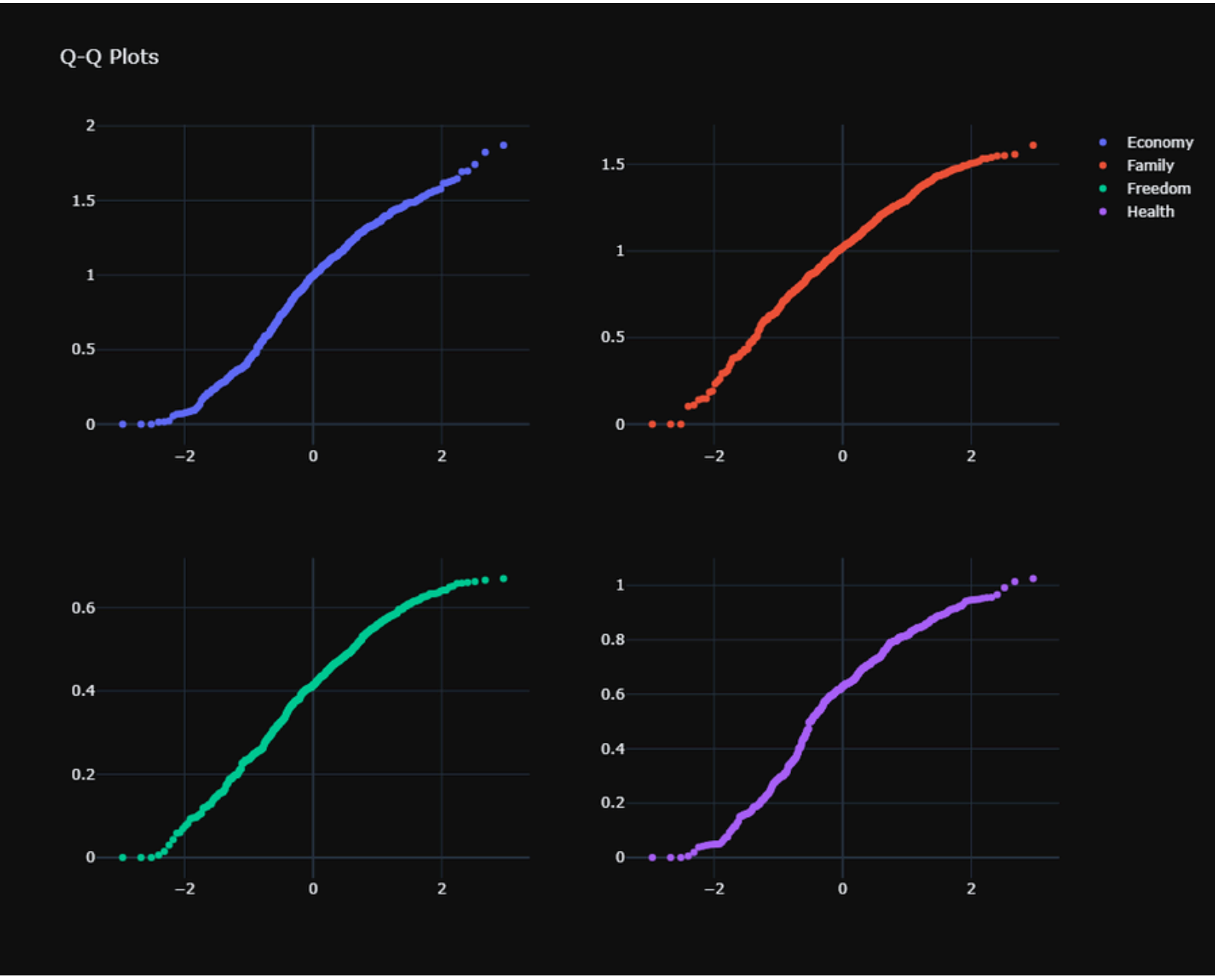
Removing outliers using LQR method

```
min=happiness.Trust.min()
q1=happiness.Trust.quantile(0.25)
q3=happiness.Trust.quantile(0.75)
IQR=q3-q1
LB=q1-(1.5*IQR)
UB=q3+(1.5*IQR)
print(LB)
print(UB)
```

Q-Q PLOT TO CHECK NORMALITY

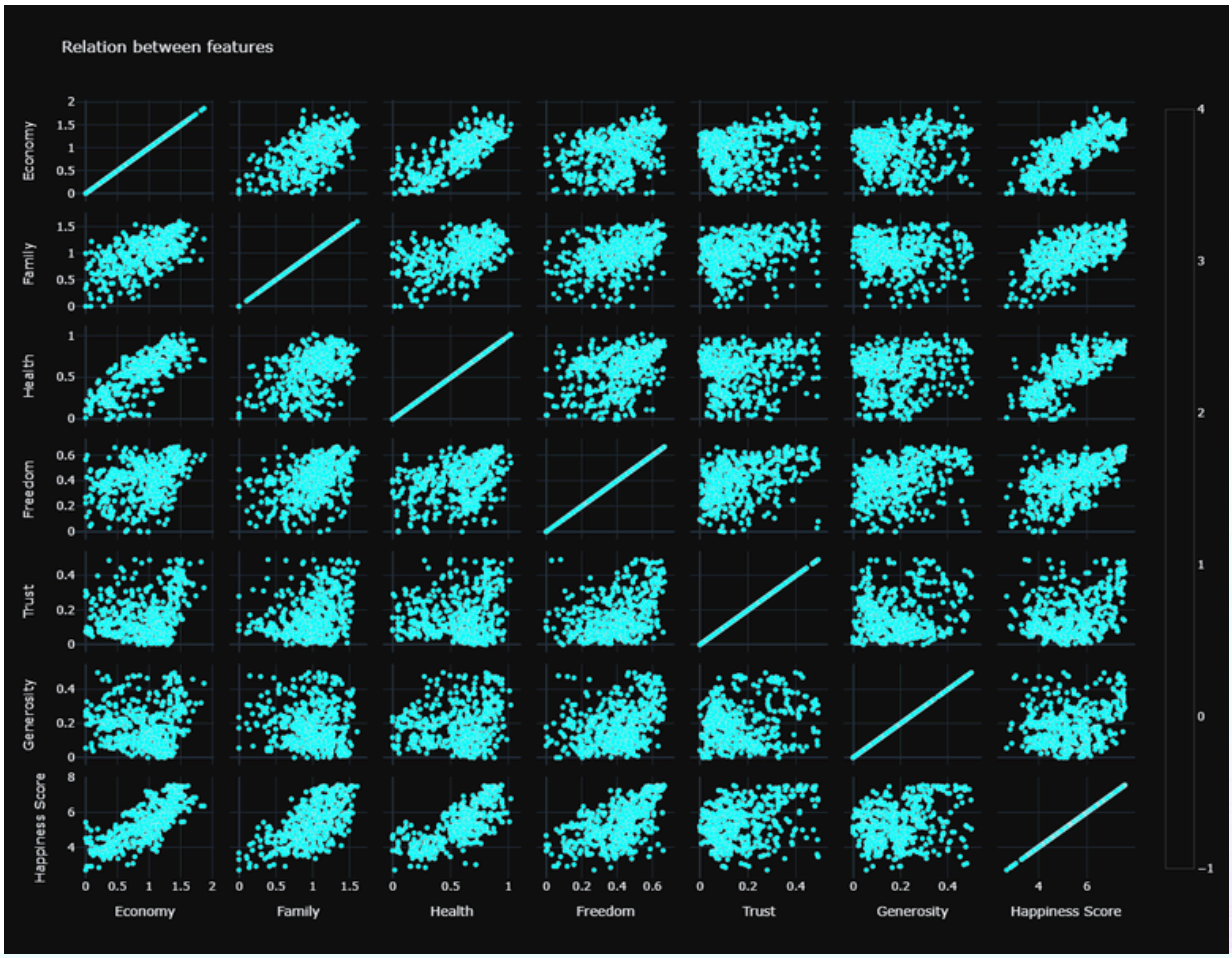
```
# Q-Q plot to check the Normality
fig = make_subplots(rows=2, cols=2)

fig.add_trace(go.Scatter(x=stats.probplot(happiness['Economy'])[0][0],
                         y=stats.probplot(happiness['Economy'])[0][1],
                         mode='markers', name='Economy'), row=1, col=1)

fig.add_trace(go.Scatter(x=stats.probplot(happiness['Family'])[0][0],
                         y=stats.probplot(happiness['Family'])[0][1],
                         mode='markers', name='Family'), row=1, col=2)

fig.add_trace(go.Scatter(x=stats.probplot(happiness['Freedom'])[0][0],
                         y=stats.probplot(happiness['Freedom'])[0][1],
                         mode='markers', name='Freedom'), row=2, col=1)

fig.add_trace(go.Scatter(x=stats.probplot(happiness['Health'])[0][0],
                         y=stats.probplot(happiness['Health'])[0][1],
                         mode='markers', name='Health'), row=2, col=2)

fig.update_layout(title='Q-Q Plots',width=1000,
height=800,template='plotly_dark')

fig.show()
```

Q-Q PLOT VISUALISATION



Visualisation between relations between features

```
fig = go.Figure(data=go.Splom(
                dimensions=[
                    dict(label='Economy', values=happiness['Economy']),
                    dict(label='Family', values=happiness['Family']),
                    dict(label='Health', values=happiness['Health']),
                    dict(label='Freedom', values=happiness['Freedom']),
                    dict(label='Trust', values=happiness['Trust']),
                    dict(label='Generosity', values=happiness['Generosity']),
                    dict(label='Happiness Score',
values=happiness['Happiness_Score'])
                ],
                showupperhalf=True,
                marker=dict(color='cyan',
                            showscale=True,
                            size=5,
                            line=dict(width=0.5, color='rgb(230,230,230)'))
                ))

fig.update_layout(title='Relation between features',
                  width=1300, height=1000,template='plotly_dark')

fig.show()
```



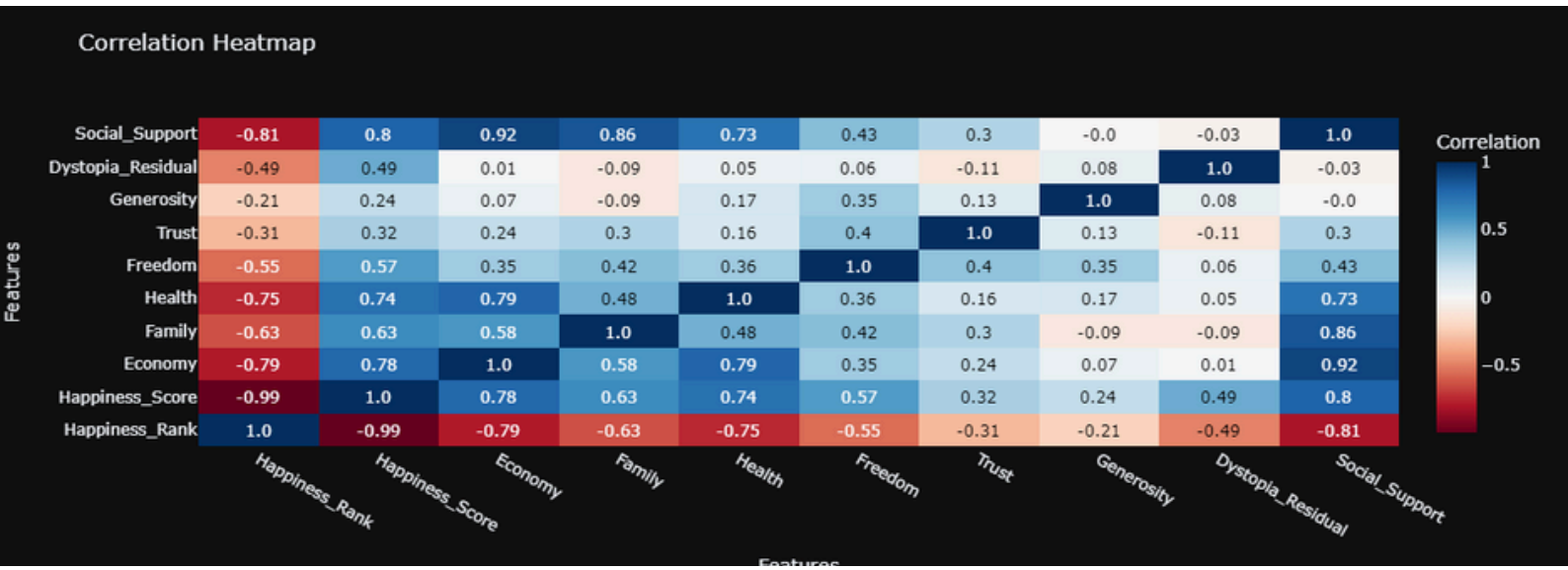
correlations between features and target

```python
target = 'Happiness_Score'

correlations = {}

for column in data_cleaned.columns:
    if column != target:
        corr = data_cleaned[column].corr(data_cleaned[target])
        correlations[column] = corr


for key, value in correlations.items():
    print(f"Correlation between {key} and {target}: {value}")
```

```
Correlation between Happiness_Rank and Happiness_Score: -0.9931274893344579
Correlation between Economy and Happiness_Score: 0.7818729853258476
Correlation between Family and Happiness_Score: 0.6296595658440398
Correlation between Health and Happiness_Score: 0.7440947592320886
Correlation between Freedom and Happiness_Score: 0.5663487557887547
Correlation between Trust and Happiness_Score: 0.3244405479973357
Correlation between Generosity and Happiness_Score: 0.241894141164536
Correlation between Dystopia_Residual and Happiness_Score: 0.4929156261134664
Correlation between Social_Support and Happiness_Score: 0.8029377865508027
```

correlation matrix

```
corr_matrix = happiness.corr(numeric_only=True)

heatmap_trace = go.Heatmap(z=corr_matrix.values,
                           x=corr_matrix.columns,
                           y=corr_matrix.columns,
                           colorscale='RdBu',
                           colorbar=dict(title='Correlation'),
                           hoverinfo='text')

layout = go.Layout(title='Correlation Heatmap',
                   xaxis=dict(title='Features'),
                   yaxis=dict(title='Features'),
                   template='plotly_dark')

annotations = []
for i, row in enumerate(corr_matrix.values):
    for j, value in enumerate(row):
        annotations.append(
            dict(
                x=corr_matrix.columns[j],
                y=corr_matrix.columns[i],
                text=str(round(value, 2)),
                font=dict(color='white' if abs(value) > 0.5 else 'black'),
                showarrow=False
            )
        )

layout.update(annotations=annotations)

fig = go.Figure(data=[heatmap_trace], layout=layout)

fig.show()
```



as we see the correlation heatmap the correlation between ECONOMY and HAPPINES_SCORE very high we get that the economy is the most important feature in our dataset

after correlation heatmap we chose the lowest correlation and drop the unimportant features

features selection

```
model = LinearRegression()

r_squared_values = []
r_squared_adjusted_values = []

for feature in X.columns:
    model.fit(X[[feature]], y)

    y_pred_train = model.predict(X[[feature]])

    r_squared = r2_score(y, y_pred_train)

    num_features = 1

    num_samples = len(X)

    r_squared_adjusted = 1 - (1 - r_squared) * (num_samples - 1) /
(num_samples - num_features - 1)

    r_squared_values.append(r_squared)
    r_squared_adjusted_values.append(r_squared_adjusted)

for feature, r_squared, r_squared_adjusted in zip(X.columns, r_squared_values,
r_squared_adjusted_values):
    print(f"Feature: {feature}, R-squared: {r_squared}, R-squared adjusted:
{r_squared_adjusted}")
```

features selection

```
Feature: Economy, R-squared: 0.6113253651823527, R-squared adjusted:
0.6104616437716468
Feature: Family, R-squared: 0.39647116885890454, R-squared adjusted:
0.39512999367859103
Feature: Health, R-squared: 0.55367701071666, R-squared adjusted:
0.5526851818515859
Feature: Freedom, R-squared: 0.3207509131834704, R-squared adjusted:
0.31924147076832254
Feature: Trust, R-squared: 0.10526166918481139, R-squared adjusted:
0.10327336178299995
Feature: Generosity, R-squared: 0.058512775529728334, R-squared adjusted:
0.056420581697572136
Feature: Social_Support, R-squared: 0.6447090890711025, R-squared adjusted:
0.6439195537134828
```

R-squared and adjusted R-squared values are approximately the same which means:

1. All Features Are Equally Important: It's possible that all the features in your model are equally important in explaining the variance in the target variable.

2. High Correlation Among Features: If the features in your model are highly correlated with each other, adding more features may not substantially improve the model's explanatory power. As a result, both R2 and R2_adj will be similar because the penalty term in R2_adj won't significantly affect the adjustment.

3. No Overfitting

Feature Engineering

```
happiness['Social_Support'] = happiness['Family'] + happiness['Economy']

print("After feature engineering:")
happiness.head()
```

we added a new feature called social_support by adding family feature and economy feature to enhance our model accurecy

and this our dataset after Feature Engineering

	Country	Happiness_Rank	Happiness_Score	Economy	Family	Health	Freedom	Trust	Generosity	Dystopia_Residual	Social_Support
0	Switzerland	1	7.587	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738	2.74602
1	Iceland	2	7.561	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201	2.70455
2	Denmark	3	7.527	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204	2.68606
3	Norway	4	7.522	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531	2.78995
4	Canada	5	7.427	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176	2.64890

build model

in our project we used 8 models
we will talk about the individually

1. Polynomial Regression:

- **Characteristics:**

Polynomial regression extends linear regression by fitting a polynomial equation to the data instead of a straight line.
It can capture nonlinear relationships between the independent and dependent variables.
The degree of the polynomial determines the complexity of the model.

- **Use Cases:**

Polynomial regression is useful when the relationship between the independent and dependent variables is curvilinear or nonlinear.
It can be applied in various fields such as physics, biology, finance, and engineering to model complex relationships.

- **Features:**

Flexibility to capture nonlinear patterns.
Prone to overfitting with higher degree polynomials.
Requires careful selection of the polynomial degree.

build model

in our project we used 8 models
we will talk about the individually

2. SVR (Support Vector Regression):

- **Characteristics:**

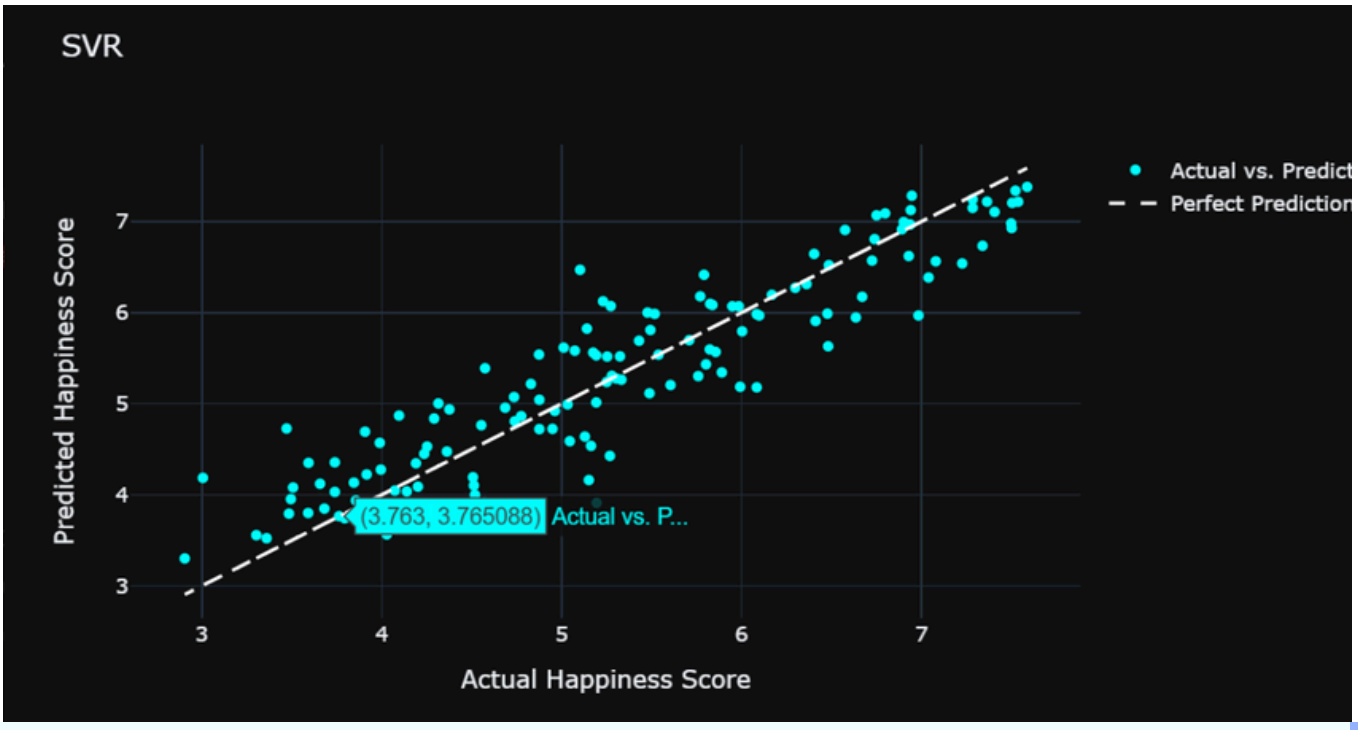
SVR is a variant of support vector machines (SVM) adapted for regression tasks.
It uses support vectors to define a hyperplane that maximizes the margin while minimizing the error.
SVR is effective in high-dimensional spaces and is robust to outliers.

- **Use Cases:**

SVR is suitable for datasets with complex relationships and a large number of features.
It is commonly used in financial forecasting, time series prediction, and stock market analysis.

- **Features:**

Capable of capturing complex nonlinear relationships.
Robust to outliers due to the use of epsilon-insensitive loss function.
Requires tuning of hyperparameters such as kernel type, regularization parameter (C), and kernel coefficient (gamma).



build model

in our project we used 8 models
we will talk about the individually

3. Linear Regression:

- **Characteristics:**

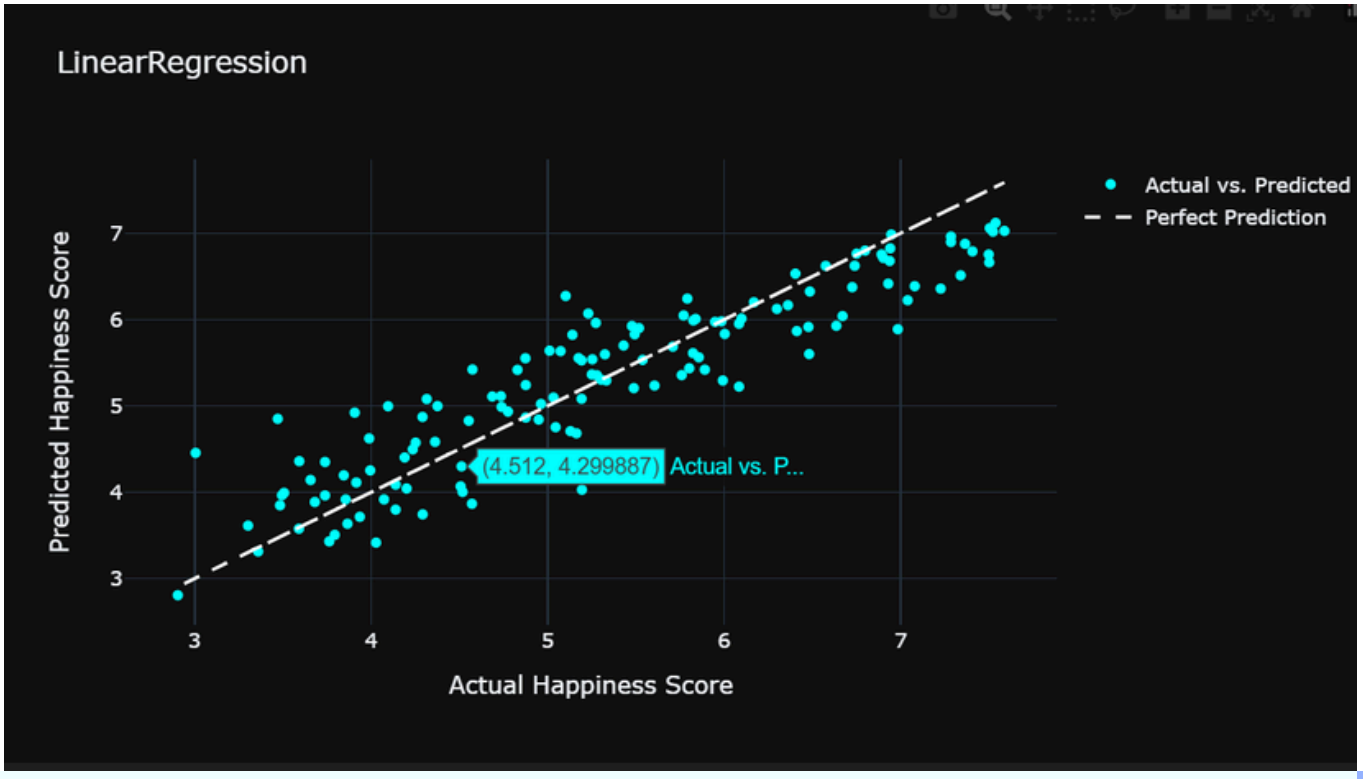
Linear regression models the relationship between the independent and dependent variables using a linear equation.
It assumes a linear relationship between the features and the target variable.
Least squares method is commonly used to estimate the coefficients.

- **Use Cases:**

Linear regression is appropriate when the relationship between the variables is approximately linear.
It is widely used in fields such as economics, social sciences, and business analytics for forecasting and inference.

- **Features:**

Simple and interpretable model.
Fast training and prediction.
Sensitive to outliers and multicollinearity.



build model

in our project we used 8 models
we will talk about the individually

4. Huber Regression:

- Characteristics:

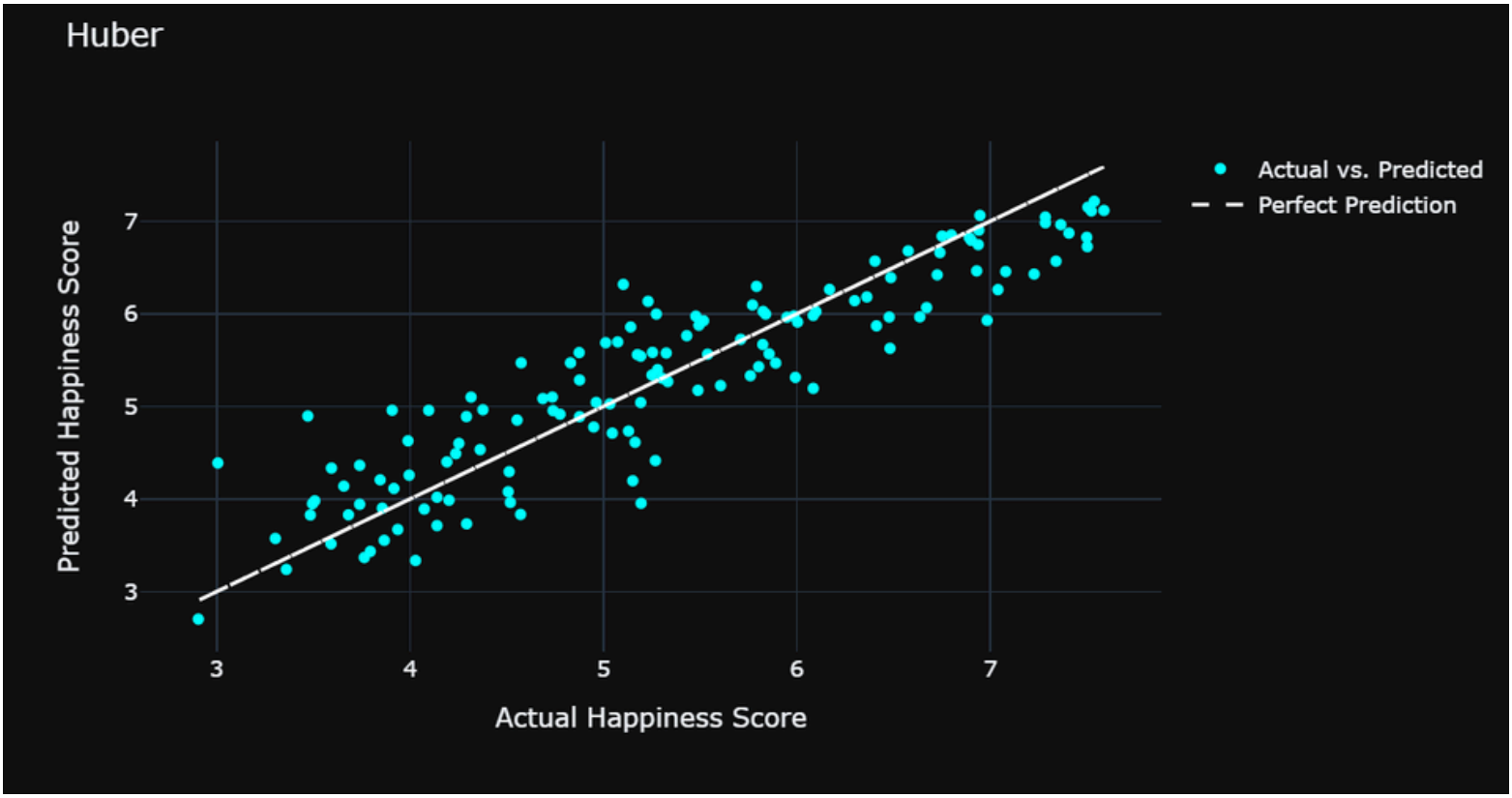
Huber regression is a robust regression technique that combines the advantages of least squares and absolute error regression.
It minimizes the sum of squared errors for small residuals and the absolute error for large residuals.
Huber loss function is less sensitive to outliers compared to least squares.

- Use Cases:

Huber regression is suitable for datasets with outliers or heavy-tailed distributions.
It is commonly used in robust statistics and data analysis where the data may contain anomalies.

- Features:

Robust to outliers and data contamination.
Balances between the advantages of least squares and absolute error regression.
Requires tuning of the delta parameter to control the transition point between squared and absolute error.



build model

in our project we used 8 models
we will talk about the individually

5. RANSAC Regression:

- **Characteristics:**

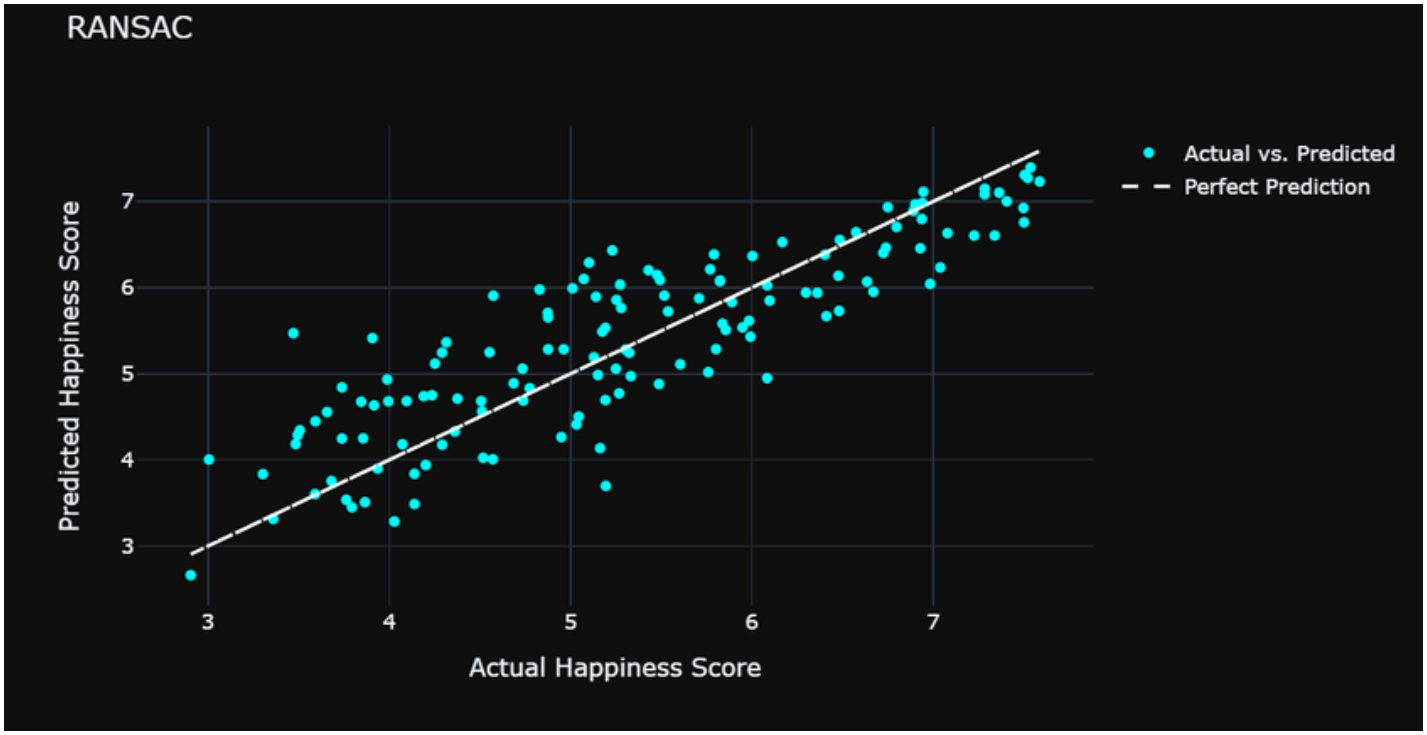
RANSAC (Random Sample Consensus) regression is a robust regression algorithm that fits a model to the inliers in the dataset.
It iteratively selects random subsets of data points, fits a model, and evaluates its performance on the entire dataset.
RANSAC is effective in the presence of outliers and noise.

- **Use Cases:**

RANSAC regression is useful for datasets contaminated with outliers or erroneous data points.
It is commonly used in computer vision, image processing, and geometric modeling to estimate transformation parameters robustly.

- **Features:**

Robust to outliers and noisy data.
Automatically detects and removes outliers during model fitting.
May require tuning of hyperparameters such as the maximum number of iterations and the threshold for defining inliers.



build model

in our project we used 8 models
we will talk about the individually

6. Bayesian Ridge Regression:

- **Characteristics:**

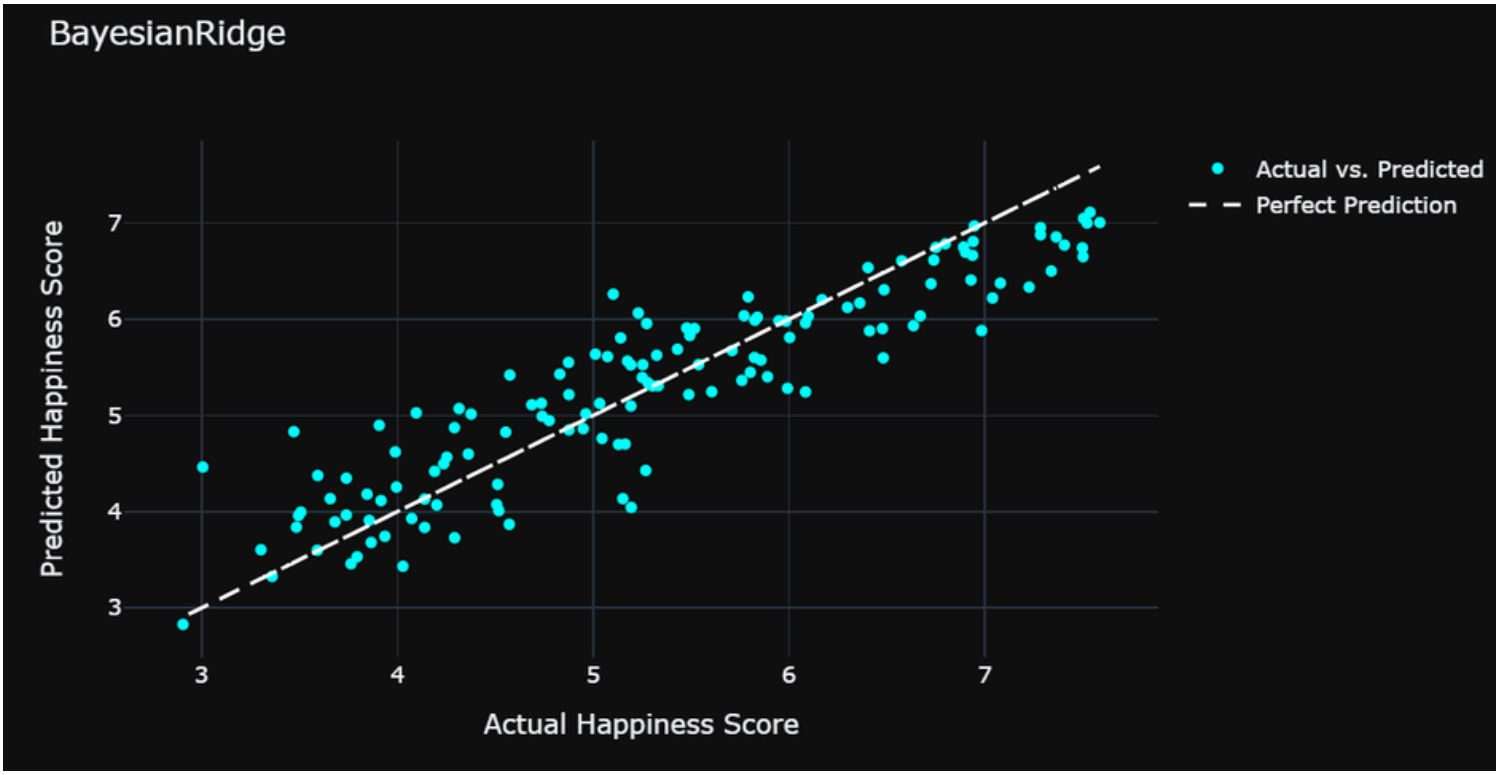
Bayesian ridge regression is a linear regression technique that incorporates Bayesian priors on the regression coefficients.
It regularizes the model by adding a penalty term based on the prior distribution of the coefficients.
Bayesian ridge regression provides a probabilistic framework for estimating uncertainty in the model parameters.

- **Use Cases:**

Bayesian ridge regression is suitable for situations where there is limited data or where regularization is necessary to prevent overfitting.
It is commonly used in Bayesian statistics, machine learning, and predictive modeling tasks.

- **Features:**

Provides uncertainty estimates for the regression coefficients.
Robust to multicollinearity and overfitting.
Requires specifying prior distributions for the regression coefficients.



build model

in our project we used 8 models
we will talk about the individually

7. Theil-Sen Regression:

- **Characteristics:**

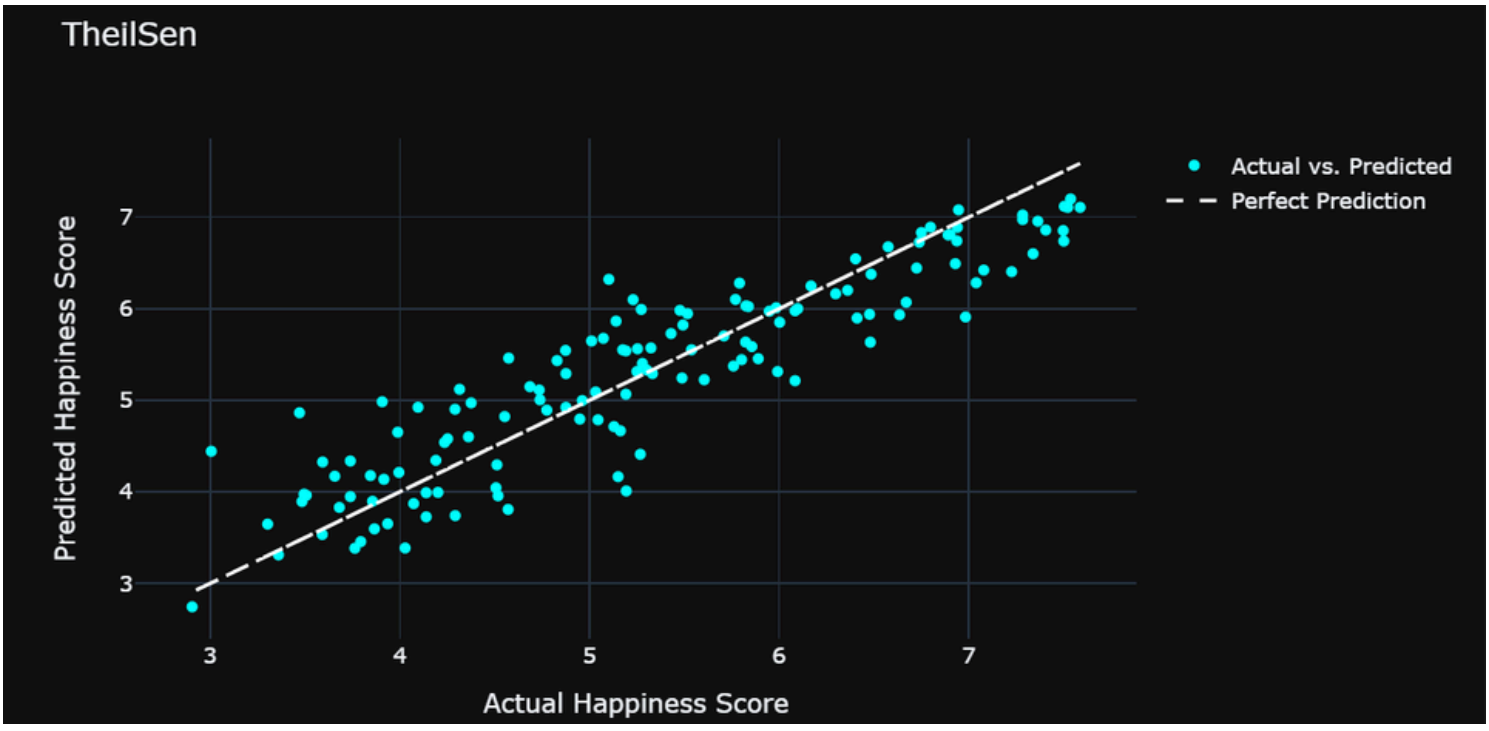
Theil-Sen regression is a nonparametric regression technique that estimates the slope of the regression line using the median of slopes between pairs of data points.
It is robust to outliers and does not assume any specific distribution for the data.
Theil-Sen regression provides a robust estimate of the slope even in the presence of outliers.

- **Use Cases:**

Theil-Sen regression is useful for datasets with outliers or non-normal distributions.
It is commonly used in environmental sciences, geology, and hydrology for trend analysis and modeling.

- **Features:**

Robust to outliers and non-normality.
Does not require specifying a distribution for the data.
Computationally efficient for large datasets.



build model

in our project we used 8 models
we will talk about the individually

8. Ridge Regression:

- **Characteristics:**

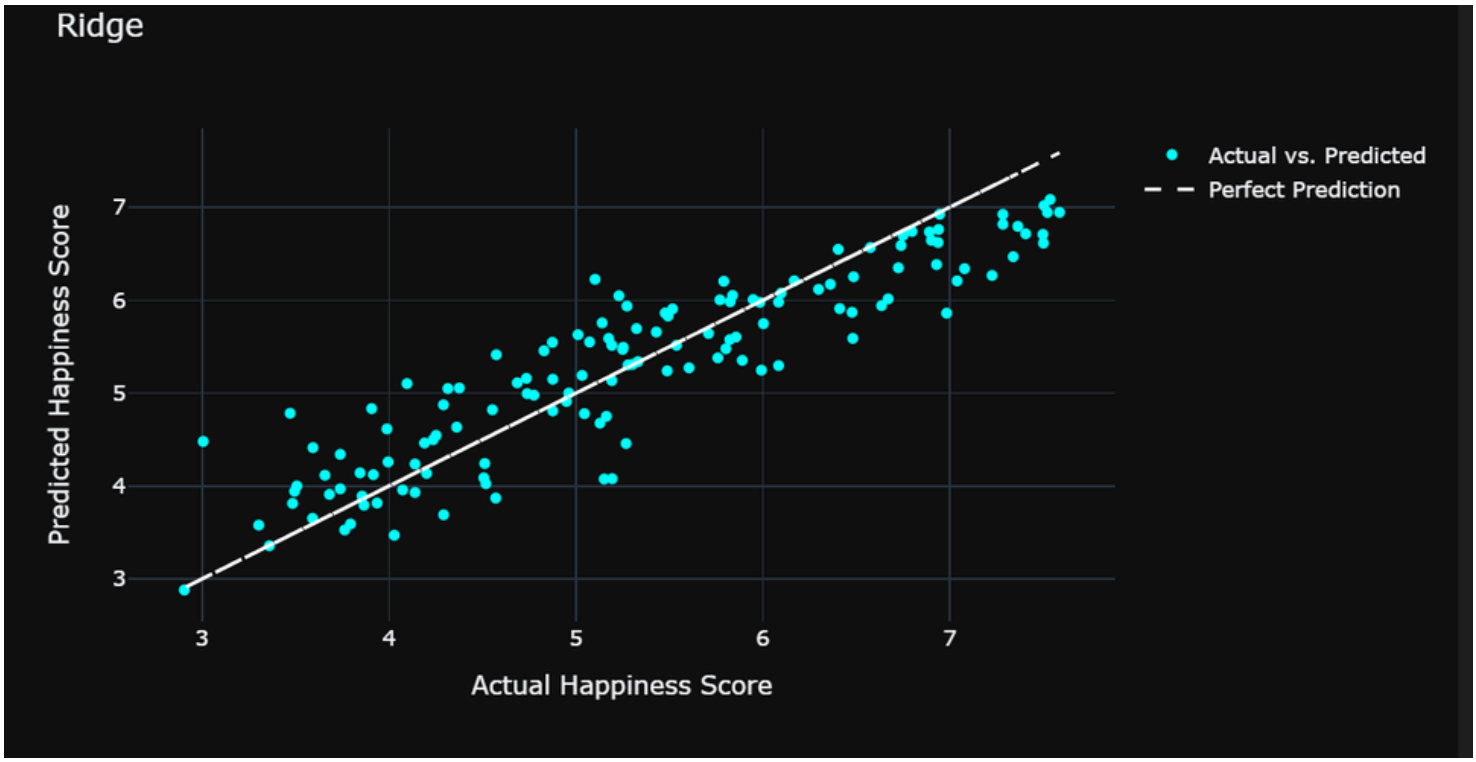
Ridge regression is a linear regression technique that adds a penalty term to the least squares objective function to regularize the model.
It shrinks the regression coefficients towards zero, reducing their variance and improving the model's generalization performance.
Ridge regression is effective in preventing overfitting, especially when the dataset has multicollinearity.

- **Use Cases:**

Ridge regression is suitable for datasets with multicollinearity or high-dimensional features.
It is commonly used in genetics, finance, and machine learning where feature selection and regularization are important.

- **Features:**

Controls overfitting by shrinking the regression coefficients.
Handles multicollinearity by stabilizing the inverse matrix.
Requires tuning of the regularization parameter (alpha) to balance between bias and variance.



build model

in our project we used 8 models
we will talk about the individually

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model_names = [
    'SVR',
    'LinearRegression',
    'Huber',
    'RANSAC',
    'BayesianRidge',
    'TheilSen',
    'Ridge'
]


models = [
    SVR(degree=7, kernel='rbf', C=10, gamma=0.1),
    LinearRegression(),
    HuberRegressor(alpha=0.0001, max_iter=10000),
    RANSACRegressor(),
    BayesianRidge(alpha_1=0.1, lambda_1=0.1),
    TheilSenRegressor(random_state=0),
    Ridge(alpha=1.0),
]

models_results = []
model_predict_results = []
for model in models:
    fitted_model = model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    model_predict_results.append(y_pred)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2e = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    models_results.append([r2,mae,mse,r2e])
```

build model

in our project we used 8 models
we will talk about the individually

```
models_df = pd.DataFrame({
    'Model':model_names,
    'Coefficient of determination (R2)':[model[0] for model in
models_results],
    'Mean Absolute Error':[model[1] for model in models_results],
    'Mean Square Error':[model[2] for model in models_results],
    'Root Mean Square Error':[model[3] for model in models_results],
})
models_df
```

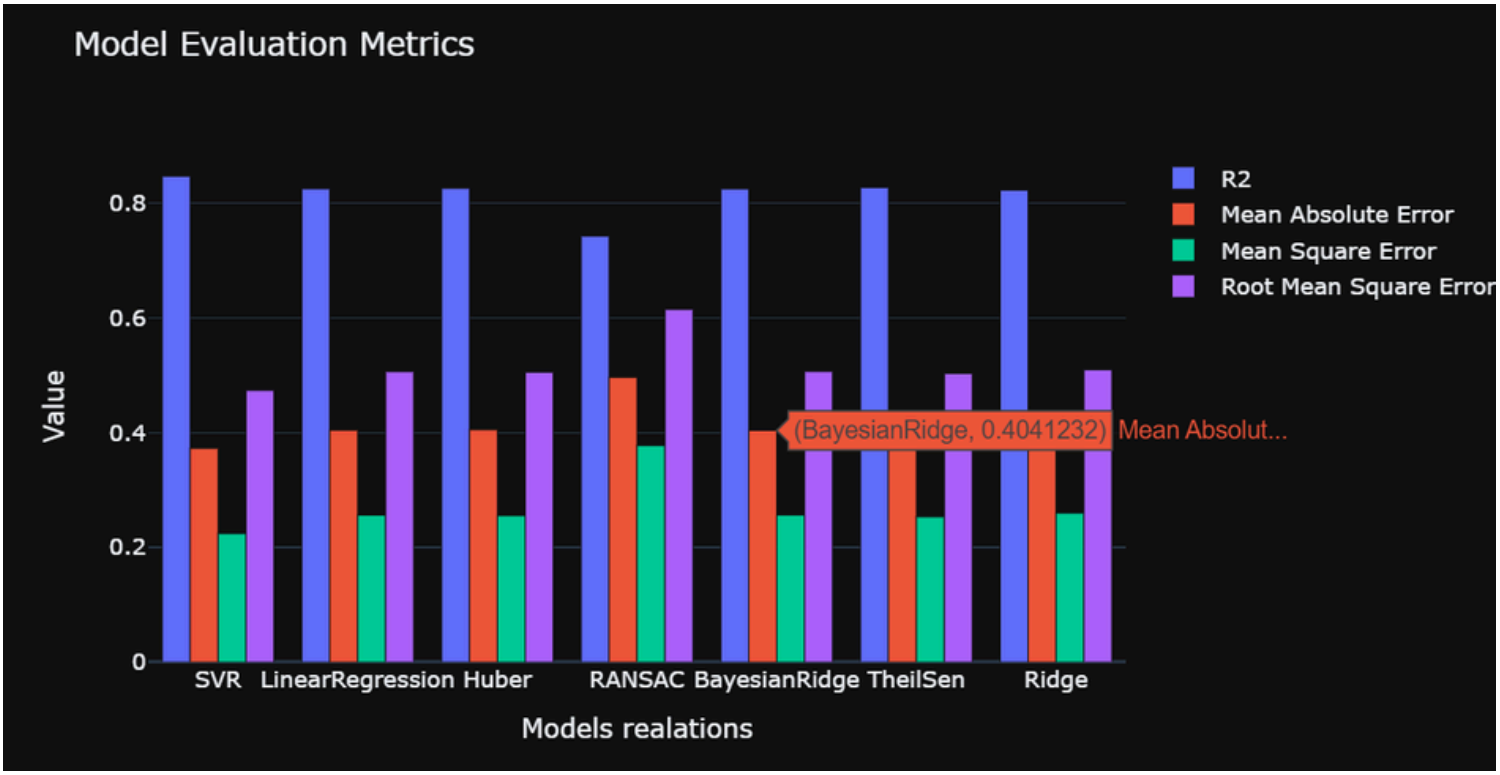
	Model	Coefficient of determination (R2)	Mean Absolute Error	Mean Square Error	Root Mean Square Error
0	SVR	0.847073	0.372728	0.224437	0.473748
1	LinearRegression	0.825248	0.404607	0.256468	0.506427
2	Huber	0.825851	0.405323	0.255583	0.505553
3	RANSAC	0.742511	0.496618	0.377894	0.614731
4	BayesianRidge	0.825032	0.404123	0.256786	0.506741
5	TheilSen	0.827339	0.403150	0.253400	0.503388
6	Ridge	0.822899	0.406087	0.259916	0.509820

these are ore models that we used in
our project and their characteristics
as we observe the R2 in every model is
approximately the same so our dataset
doesn't have overfitting

model evaluation

we compared between R2, MAE,MSE and RMSE among the models we used

```
trace_r2 = go.Bar(x=models_df['Model'], y=models_df['Coefficient of determination (R2)'], name='R2')
trace_mae = go.Bar(x=models_df['Model'], y=models_df['Mean Absolute Error'], name='Mean Absolute
Error')
trace_mse = go.Bar(x=models_df['Model'], y=models_df['Mean Square Error'], name='Mean Square Error')
trace_rmse = go.Bar(x=models_df['Model'], y=models_df['Root Mean Square Error'], name='Root Mean Square
Error')


data = [trace_r2, trace_mae, trace_mse, trace_rmse]

layout = go.Layout(title='Model Evaluation Metrics',
                   xaxis=dict(title='Models realations',tickangle=0),
                   yaxis=dict(title='Value'), template="plotly_dark")

fig = go.Figure(data=data, layout=layout)

fig.show()
```



actual vs predicted data

we tested our prediction on random data from our dataset to make sure from accuracy model we observed that the actual data and the predicted data are approximately the same

```
reg_model_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred})
reg_model_diff
```

	Actual value	Predicted value
154	3.303	3.579263
43	6.003	5.747975
37	6.168	6.209124
33	6.403	6.547041
2	7.501	6.614069
...	...	...
31	6.485	6.252150
155	3.006	4.479817
119	4.362	4.633748
28	6.575	6.566635
7	7.364	6.794329

overfitting test

```python
train_rmse = []
test_rmse = []

for model in models:
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    train_rmse.append(np.sqrt(mean_squared_error(y_train, y_train_pred)))
    test_rmse.append(np.sqrt(mean_squared_error(y_test, y_test_pred)))

# for i, model_name in enumerate(model_names):
#     print("Model: {}, Train RMSE: {:.2f}, Test RMSE: {:.2f}".format(model_name, train_rmse[i],
test_rmse[i]))

pd.DataFrame({
    'Models': model_names,
    'Train RMSE': [result for result in train_rmse],
    'Test RMSE': [result for result in test_rmse],
})
```

- **RMSE** (Root Mean Squared Error) measures how well a model predicts on a dataset. A lower RMSE indicates a better fit.
- **Training data** is used to train the model. The model learns patterns from this data.
- **Test data** is unseen data that the model has not been exposed to during training. It's used to evaluate how well the model generalizes to unseen data.

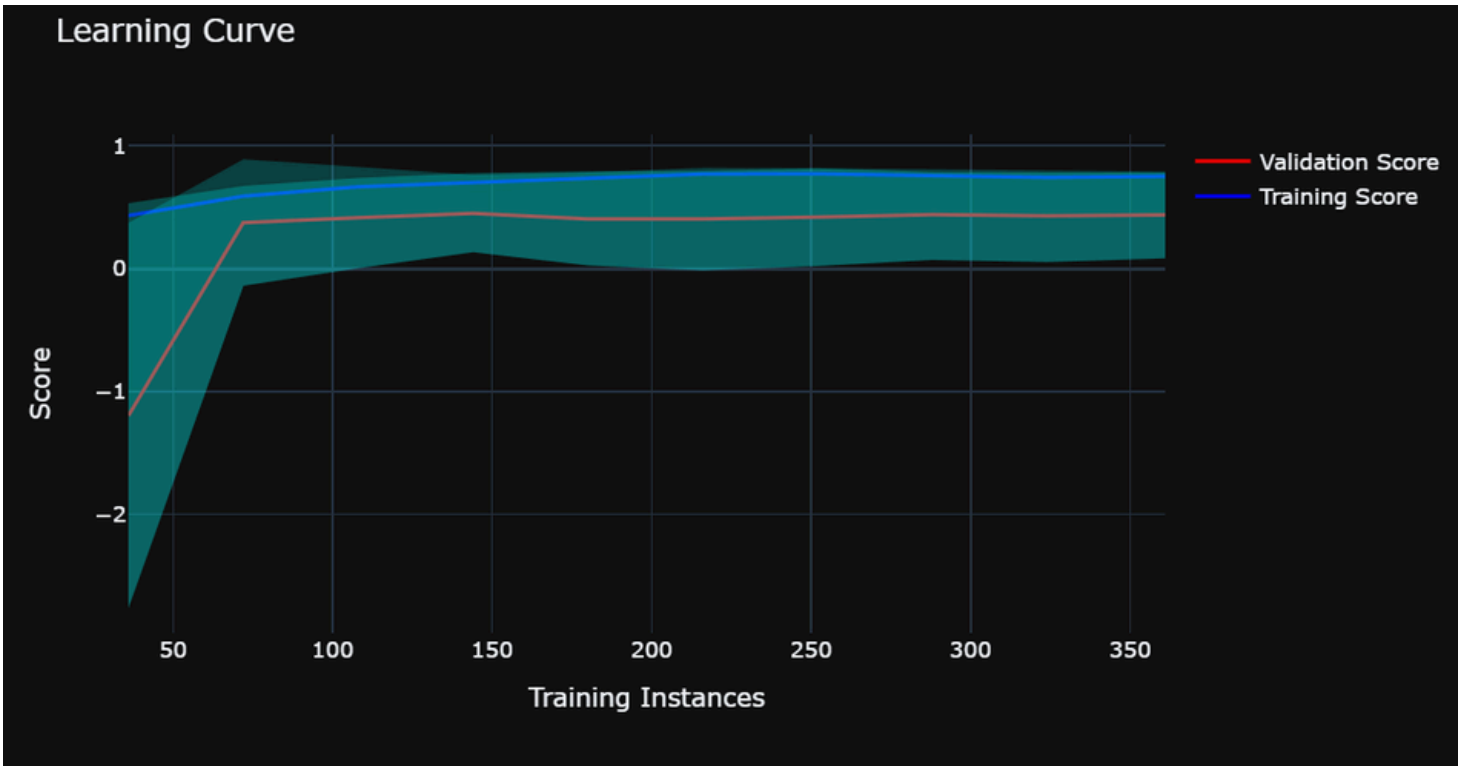
	Models	Train RMSE	Test RMSE
0	SVR	0.574816	0.473748
1	LinearRegression	0.587654	0.506427
2	Huber	0.589160	0.505553
3	RANSAC	0.609342	0.520743
4	BayesianRidge	0.587790	0.506741
5	TheilSen	0.589448	0.503388
6	Ridge	0.589383	0.509820

Overfitting occurs when a model memorizes the training data too well, capturing noise and irrelevant details. This leads to a low RMSE on the training data. However, when applied to unseen test data, the model performs poorly because it hasn't learned the underlying patterns effectively.
This results in a higher **RMSE** on the **test data** compared to the **training data**.
and the vice versa for **underfitting**

learning curve

```
def plot_learning_curve(estimator, X, y):
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, train_sizes=np.linspace(0.1, 1.0, 10), cv=5)

    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)

    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    trace_train = go.Scatter(x=train_sizes, y=train_mean, mode='lines', name='Training Score',
line=dict(color='blue'))
    trace_test = go.Scatter(x=train_sizes, y=test_mean, mode='lines', name='Validation Score',
line=dict(color='red'))

    fig = go.Figure([trace_train, trace_test])
    fig.add_trace(go.Scatter(x=train_sizes, y=train_mean - train_std,
                             mode='lines',
                             showlegend=False,
                             line=dict(width=0),
                             fill='tonexty',
                             fillcolor='rgba(0,100,80,0.2)'))

    fig.add_trace(go.Scatter(x=train_sizes, y=train_mean + train_std,
                             mode='lines',
                             showlegend=False,
                             line=dict(width=0),
                             fill='tonexty',
                             fillcolor='rgba(0,100,80,0.2)'))

    fig.add_trace(go.Scatter(x=train_sizes, y=test_mean - test_std,
                             mode='lines',
                             showlegend=False,
                             line=dict(width=0),
                             fill='tonexty',
                             fillcolor='rgba(0,255,255,0.2)'))

    fig.add_trace(go.Scatter(x=train_sizes, y=test_mean + test_std,
                             mode='lines',
                             showlegend=False,
                             line=dict(width=0),
                             fill='tonexty',
                             fillcolor='rgba(0,255,255,0.2)'))

    fig.update_layout(title='Learning Curve',
                      xaxis_title='Training Instances',
                      yaxis_title='Score',
                      hovermode='x',
                      template='plotly_dark')

    fig.show()


plot_learning_curve(poly_model, X, y)
```

learning curve



A learning curve in a regression project visually depicts how a regression model's performance changes with varying amounts of training data. It shows the relationship between the size of the training dataset and the model's error or accuracy. By analyzing the learning curve, researchers can assess if the model suffers from underfitting or overfitting and make informed decisions about improving the model's performance, such as adjusting complexity or acquiring more data.

ols model

Ordinary Least Squares (OLS) regression is a method used in statistics and machine learning to find the best-fitting line or plane through a set of data points. Here's a simple explanation:

- Line of Best Fit: Imagine you have a scatterplot with points scattered around. OLS helps you find the straight line that best represents the overall trend of the data.
- Minimizing Errors: The "least squares" part means that OLS minimizes the sum of the squared vertical distances (errors) between each data point and the line. It tries to find the line that makes these distances as small as possible.
- Finding Coefficients: OLS calculates the slope and intercept of the line that minimize the errors. These coefficients represent the relationship between the independent variable (X) and the dependent variable (Y).

Predictions: Once you have the coefficients, you can use the line to make predictions. Given a new value of X, you can plug it into the equation of the line to predict the corresponding Y value.

ols model

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

ols_model = ols('Happiness_Score ~ Economy + Family + Health + Freedom + Trust
+ Social_Support', data=data_cleaned).fit()
X1 = np.column_stack((data_cleaned['Economy'], data_cleaned['Family'],
data_cleaned['Health'], data_cleaned['Freedom'], data_cleaned['Trust'],
data_cleaned['Generosity'], data_cleaned['Social_Support']))
X2 = sm.add_constant(X1)  # Add constant term for intercept
y = data_cleaned['Happiness_Score']

model = sm.OLS(y, X2).fit()

print(model.summary())

predictions = model.predict(X2)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X1[:, 0], X1[:, 1], y, c='blue', marker='o', label='Observed')

ax.scatter(X1[:, 0], X1[:, 1], predictions, c='red', marker='^',
label='Predicted')

ax.set_xlabel('Economy')
ax.set_ylabel('Family')
ax.set_zlabel('Happiness_Score')
ax.set_title('Multiple Linear Regression')
ax.legend()

plt.show()
```

ols model

OLS Regression Results

Dep. Variable:	Happiness_Score	R-squared:	0.765
Model:	OLS	Adj. R-squared:	0.761
Method:	Least Squares	F-statistic:	240.9
Date:	Thu, 09 May 2024	Prob (F-statistic):	2.49e-136
Time:	11:26:18	Log-Likelihood:	-371.52
No. Observations:	452	AIC:	757.0
Df Residuals:	445	BIC:	785.8
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.1239	0.101	21.011	0.000	1.925	2.323
x1	0.4490	0.093	4.813	0.000	0.266	0.632
x2	0.1384	0.092	1.497	0.135	-0.043	0.320
x3	1.1727	0.182	6.437	0.000	0.815	1.531
x4	1.5440	0.223	6.938	0.000	1.107	1.981
x5	0.3382	0.241	1.401	0.162	-0.136	0.813
x6	1.0810	0.248	4.367	0.000	0.595	1.568
x7	0.5873	0.044	13.481	0.000	0.502	0.673

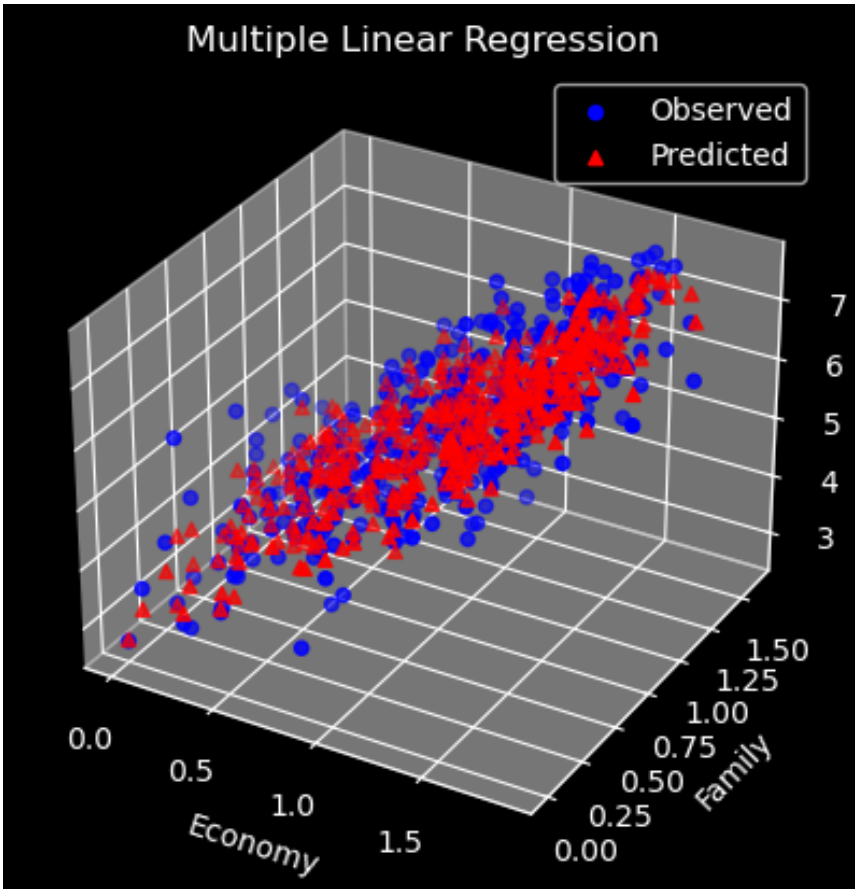
Omnibus:	8.411	Durbin-Watson:	1.481
Prob(Omnibus):	0.015	Jarque-Bera (JB):	9.665

...

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 4.36e-29. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.



anova table

An ANOVA table is a statistical summary that breaks down the variation observed in a dataset into different sources, such as between groups and within groups. It includes components like sums of squares, degrees of freedom, mean squares, F-values, and p-values. These elements help determine the significance of differences between group means, providing insights into the factors affecting variation in the data.

```
ssr = ols_model.mse_model * ols_model.df_model
sse = ols_model.mse_resid * ols_model.df_resid

df_reg = ols_model.df_model
df_error = ols_model.df_resid

msr = ssr / df_reg
mse = sse / df_error

f_statistic = msr / mse

anova_table = sm.stats.anova_lm(ols_model, typ=2)

anova_table['PR(>F)'] = anova_table['PR(>F)'].apply(lambda x:
'{:.15f}'.format(x))
print(anova_table)
```

```
                    sum_sq     df          F                PR(>F)
Economy           8.155436    1.0  25.465923  0.000000656828329
Family            0.102470    1.0   0.319969  0.571910843312174
Health           16.282872    1.0  50.844411  0.000000000004059
Freedom          26.520111    1.0  82.810912  0.000000000000000
Trust             0.777968    1.0   2.429261  0.119798332068794
Social_Support   50.114488    1.0 156.486012  0.000000000000000
Residual        142.831053  446.0         NaN                nan
```

data prediction

we put some values from our independent features to predict the dependent value

```
# Predicts the happiness score for a new data point using the trained
regression model.

new_data_point = np.array([[0.91851 , 1.00232, 0.73545, 0.33457, 1.92083]])

predicted_happiness_score = models[1].predict(new_data_point)

print("Predicted Happiness Score:", predicted_happiness_score[0])

"output"
Predicted Happiness Score: 5.41116552628276
```

load a new dataset and rename columns

```
# load dataframe from csv and rename columns
happiness_2018 = pd.read_csv("../data/2018.csv")
happiness_2018.columns = ["Happiness_Rank",'Country', 'Happiness_Score',
'Economy', 'Social_support','Health',"Freedom", 'Generosity','Perceptions of
corruption']

happiness_2018.columns
happiness_2018.drop(columns="Perceptions of corruption")
common_columns = ['Country', 'Happiness_Rank', 'Happiness_Score', 'Economy',
'Health', 'Freedom', 'Generosity']

df1_filtered = happiness[common_columns]
df2_filtered = happiness_2018[common_columns]
```

data prediction

then we predict the new dataset

```
X_train = df1_filtered[['Economy', 'Health', 'Freedom', 'Generosity']]
y_train = df1_filtered['Happiness_Score']

X_test = df2_filtered[['Economy', 'Health', 'Freedom', 'Generosity']]


fitted_model = LinearRegression().fit(X_train, y_train)

y_pred = fitted_model.predict(X_test)

fitted_model.fit(X_train, y_train)

predictions = fitted_model.predict(X_test)

df2_filtered['Predicted_Happiness_Score'] = predictions

# Evaluate the test
r2 = r2_score(y_true=df2_filtered['Happiness_Score'], y_pred=predictions)
mse = mean_squared_error(y_true=df2_filtered['Happiness_Score'],
y_pred=predictions)

print("R2 Score:", r2)
print("Mean Squared Error:", mse)

"output"
R2 Score: 0.7466963010968335
Mean Squared Error: 0.31542881782409593
```

so the R2 was 0.74 and this is acceptable for performance of predicting a new dataset

thank you