ASSIGNMENT 1 MACHINE LEARNING

SYABAB ALHAQQI BIN JAAFAR 2211117

Q1 housing

A) Parameters selected is: ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income']. These are all the important parameters to ensure accurate data prediction.

Coding:

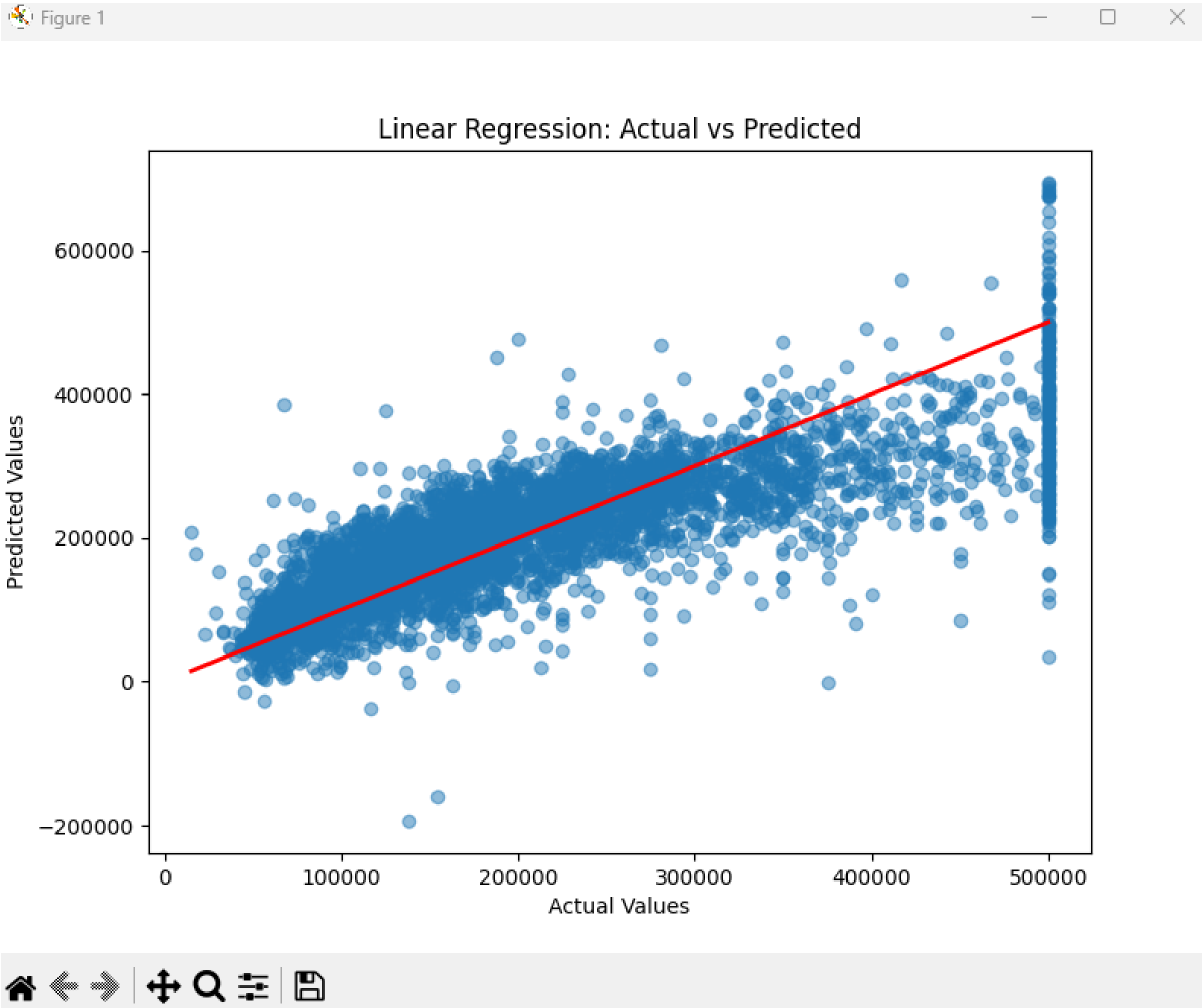
```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.linear_model import LinearRegression

from sklearn.svm import SVR

from sklearn.metrics import mean_squared_error, r2_score


# Load Dataset

df = pd.read_csv("housing.csv")


# Data Preparation

# Handle missing values

df['total_bedrooms'].fillna(df['total_bedrooms'].median(), inplace=True)


# One-Hot Encoding for categorical variable

df = pd.get_dummies(df, columns=['ocean_proximity'], drop_first=True)


# Feature Scaling

scaler = StandardScaler()

numerical_features = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',

              'total_bedrooms', 'population', 'households', 'median_income']
```

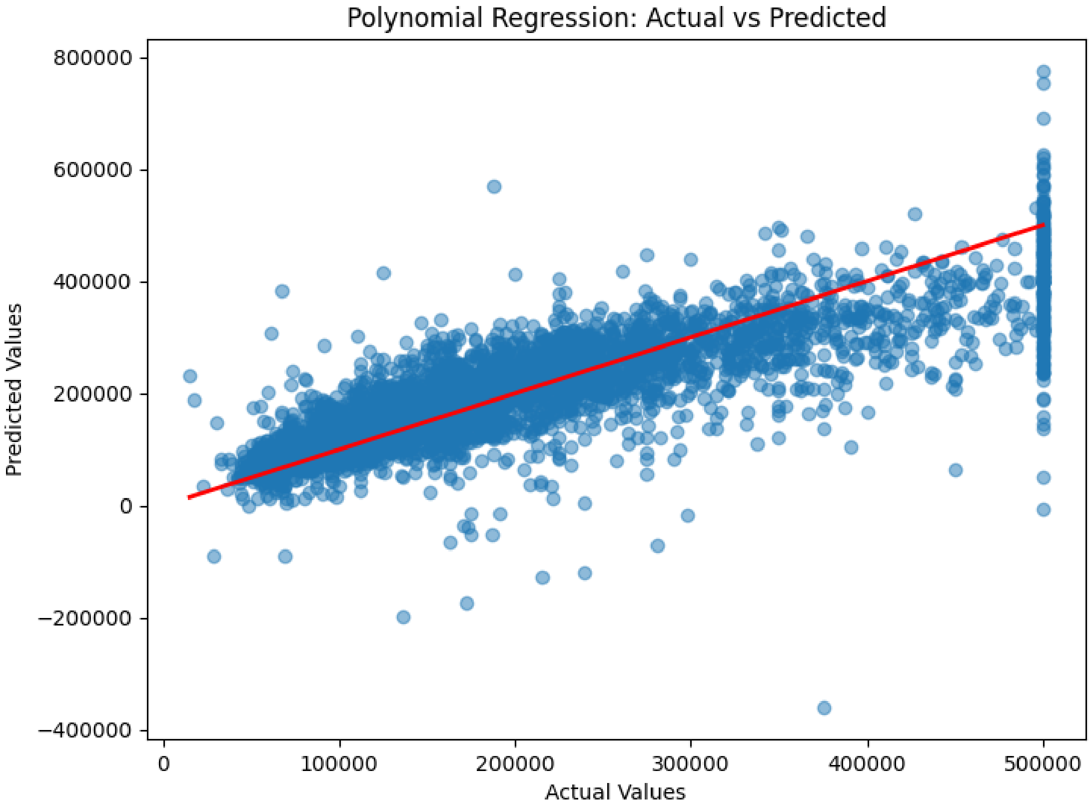
```
df[numerical_features] = scaler.fit_transform(df[numerical_features])

# Define Features and Target
X = df.drop(columns=['median_house_value'])
y = df['median_house_value']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --- Multiple Linear Regression ---
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_lr_pred = lr_model.predict(X_test)
lr_mse = mean_squared_error(y_test, y_lr_pred)
lr_r2 = r2_score(y_test, y_lr_pred)

# --- Polynomial Regression (Degree 2) ---
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)
y_poly_pred = poly_model.predict(X_test_poly)
poly_mse = mean_squared_error(y_test, y_poly_pred)
poly_r2 = r2_score(y_test, y_poly_pred)

# --- Support Vector Regression (SVR) ---
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).ravel()
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1)).ravel()
```

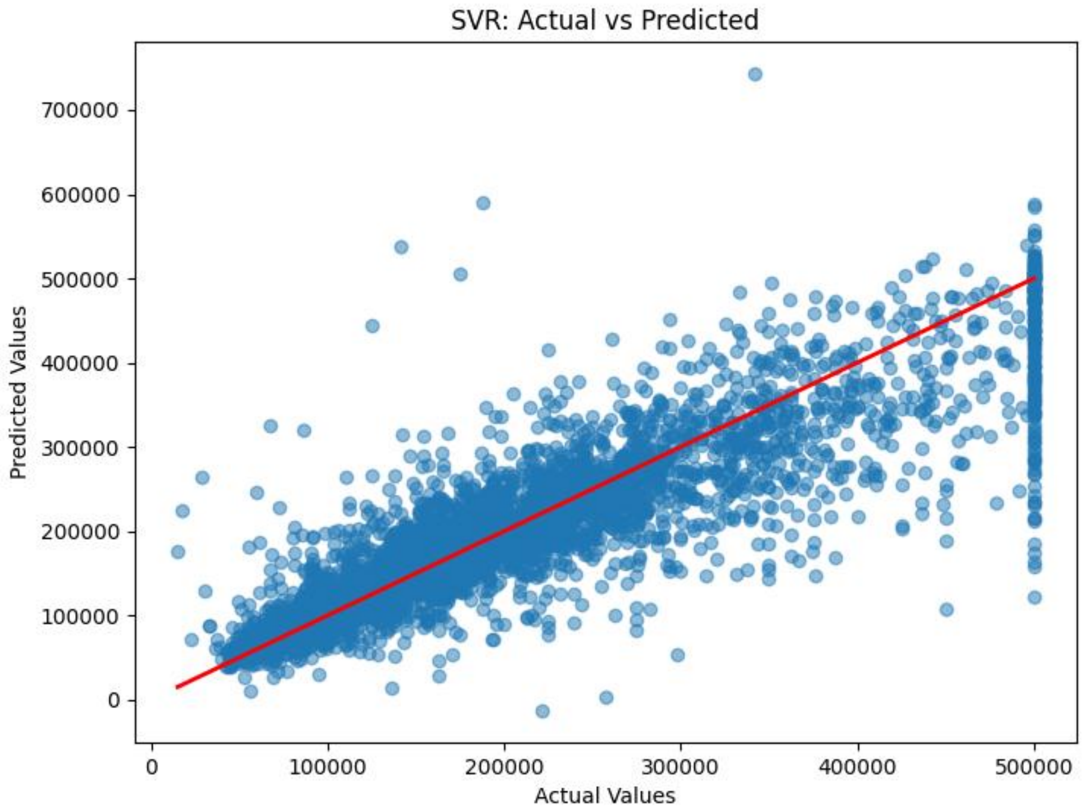
```python
svr_model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
svr_model.fit(X_train, y_train_scaled)
y_svr_pred_scaled = svr_model.predict(X_test)
y_svr_pred = scaler_y.inverse_transform(y_svr_pred_scaled.reshape(-1, 1)).ravel()
svr_mse = mean_squared_error(y_test, y_svr_pred)
svr_r2 = r2_score(y_test, y_svr_pred)

# Display results
results = pd.DataFrame({
    "Model": ["Linear Regression", "Polynomial Regression (Degree 2)", "Support Vector Regression"],
    "MSE": [lr_mse, poly_mse, svr_mse],
    "R² Score": [lr_r2, poly_r2, svr_r2]
})

print(results)

# --- Plot Predictions vs Actual Values ---
def plot_results(y_test, y_pred, title):
    plt.figure(figsize=(8, 6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r', linewidth=2)
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title(title)
    plt.show()

plot_results(y_test, y_lr_pred, "Linear Regression: Actual vs Predicted")
plot_results(y_test, y_poly_pred, "Polynomial Regression: Actual vs Predicted")
plot_results(y_test, y_svr_pred, "SVR: Actual vs Predicted")
```

F) The best model is SVR as it has the highest R^2 score and the lowest MAE/MSE.

	Model	MSE	R² Score
0	Linear Regression	4.908477e+09	0.625424
1	Polynomial Regression (Degree 2)	4.509304e+09	0.655886
2	Support Vector Regression	3.077935e+09	0.765116




Polynomial Regression: Actual vs Predicted
Predicted Values
Actual Values
800000
600000
400000
200000
0
−200000
−400000
0
100000
200000
300000
400000
500000


SVR: Actual vs Predicted
Predicted Values
Actual Values
700000
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000

Q2 solar-based energy

A) Parameters selected is: ["shortwave_radiation_backwards_sfc", "total_cloud_cover_sfc", "high_cloud_cover_high_cld_lay", "medium_cloud_cover_mid_cld_lay", "low_cloud_cover_low_cld_lay", "temperature_2_m_above_gnd", "relative_humidity_2_m_above_gnd", "mean_sea_level_pressure_MSL", "total_precipitation_sfc", "snowfall_amount_sfc", "wind_speed_10_m_above_gnd", "wind_speed_80_m_above_gnd", "wind_speed_900_mb", "wind_direction_10_m_above_gnd", "wind_direction_80_m_above_gnd", "wind_direction_900_mb", "angle_of_incidence", "zenith", "azimuth"]. These selected features are physically relevant for PV power prediction.

Coding:

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import MinMaxScaler, PolynomialFeatures

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.svm import SVR

from sklearn.pipeline import make_pipeline

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


# Load the dataset

file_path = "SolarPowerGenerationKaggle_MissingData.xlsx"

df = pd.read_excel(file_path, sheet_name="spg")


# Select relevant features

selected_features = [

    "shortwave_radiation_backwards_sfc", "total_cloud_cover_sfc",

    "high_cloud_cover_high_cld_lay", "medium_cloud_cover_mid_cld_lay",
"low_cloud_cover_low_cld_lay",

    "temperature_2_m_above_gnd", "relative_humidity_2_m_above_gnd",
"mean_sea_level_pressure_MSL",
```

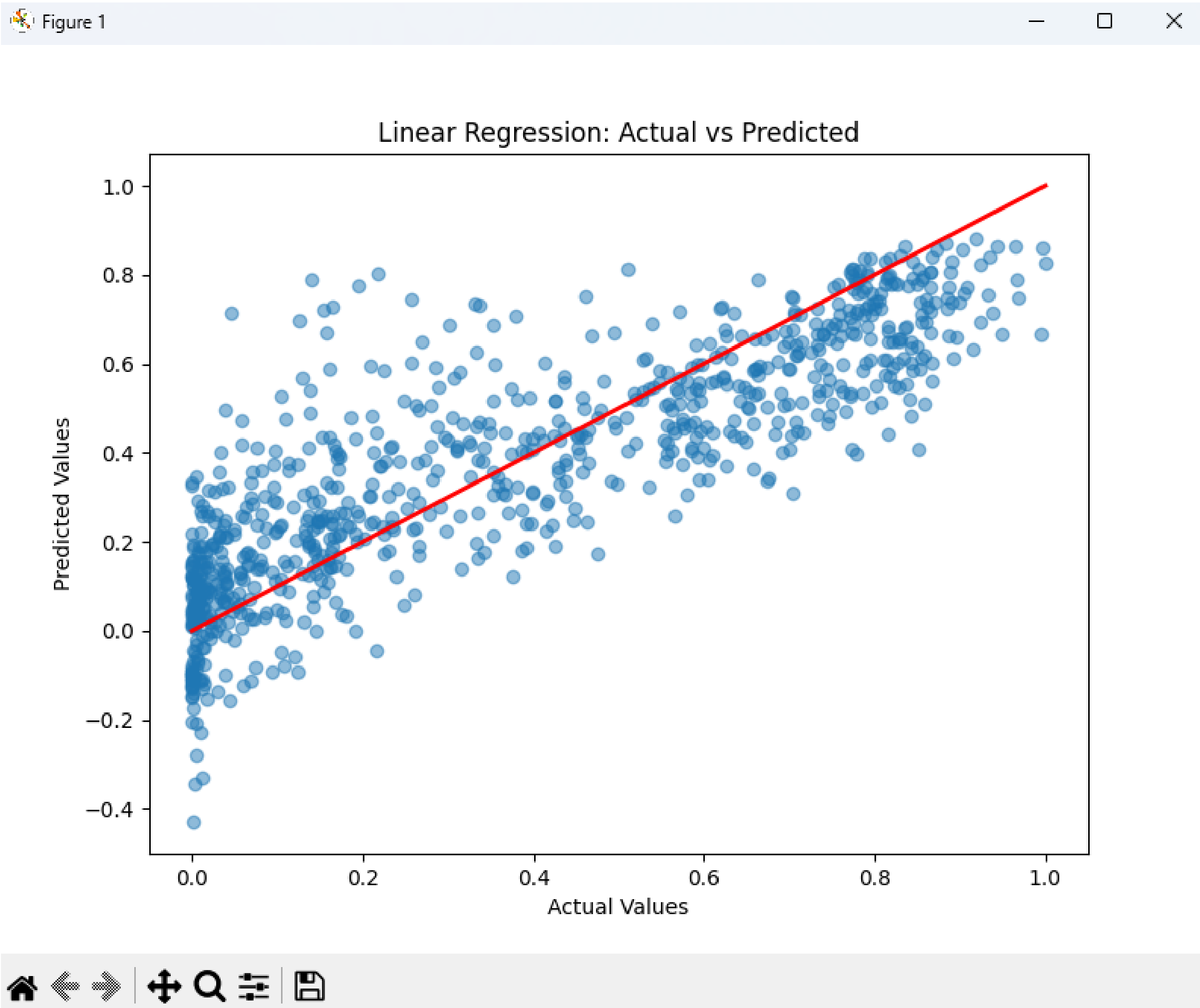
```
    "total_precipitation_sfc", "snowfall_amount_sfc", "wind_speed_10_m_above_gnd",
    "wind_speed_80_m_above_gnd", "wind_speed_900_mb",
"wind_direction_10_m_above_gnd",
    "wind_direction_80_m_above_gnd", "wind_direction_900_mb", "angle_of_incidence",
    "zenith", "azimuth"
]

df_selected = df[selected_features + ["generated_power_kw"]]

# Handle missing values with mean imputation
imputer = SimpleImputer(strategy="mean")
df_imputed = pd.DataFrame(imputer.fit_transform(df_selected),
columns=df_selected.columns)

# Scale data using Min-Max Scaling
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_imputed),
columns=df_selected.columns)

# Split data into features (X) and target variable (y)
X = df_scaled.drop(columns=["generated_power_kw"])
y = df_scaled["generated_power_kw"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --- Multiple Linear Regression ---
mlr_model = LinearRegression()
mlr_model.fit(X_train, y_train)
y_pred_mlr = mlr_model.predict(X_test)
mae_mlr = mean_absolute_error(y_test, y_pred_mlr)
mse_mlr = mean_squared_error(y_test, y_pred_mlr)
r2_mlr = r2_score(y_test, y_pred_mlr)
```

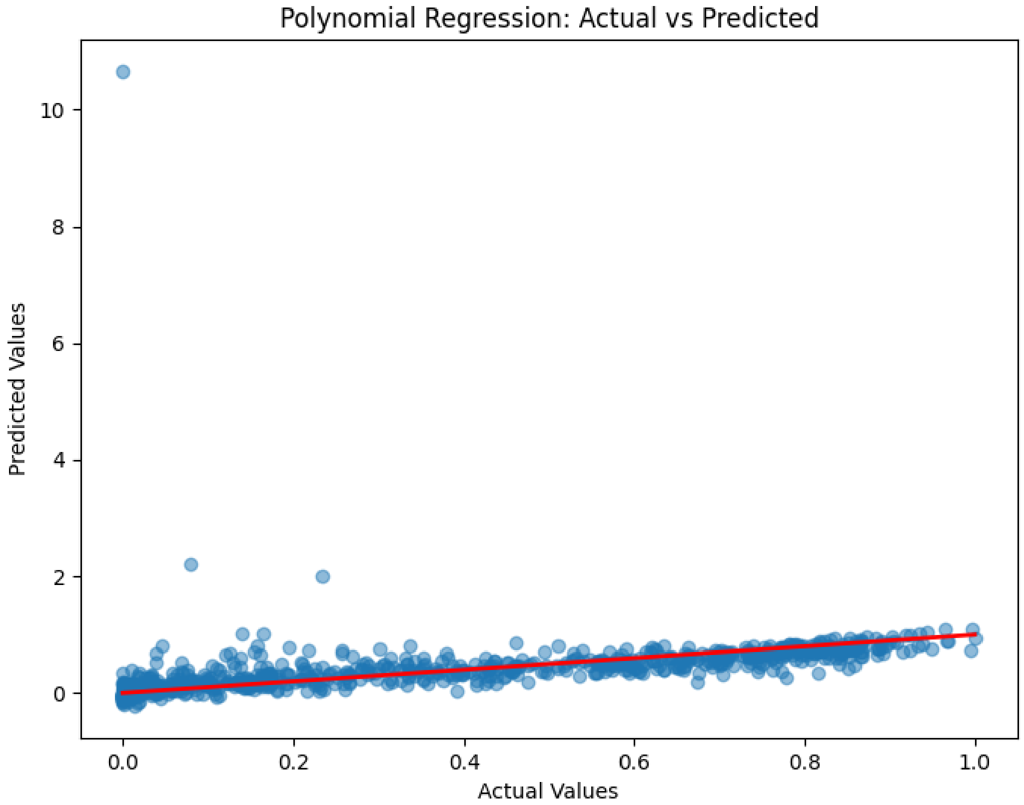
```
# --- Polynomial Regression (Degree 2) ---
poly_degree = 2
poly_model = make_pipeline(PolynomialFeatures(degree=poly_degree),
LinearRegression())
poly_model.fit(X_train, y_train)
y_pred_poly = poly_model.predict(X_test)
mae_poly = mean_absolute_error(y_test, y_pred_poly)
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

# --- Support Vector Regression (SVR) ---
svr_model = SVR(kernel='rbf')
svr_model.fit(X_train, y_train)
y_pred_svr = svr_model.predict(X_test)
mae_svr = mean_absolute_error(y_test, y_pred_svr)
mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)

# Print model evaluation results
print("Multiple Linear Regression:", f"MAE: {mae_mlr:.4f}, MSE: {mse_mlr:.4f}, R2:
{r2_mlr:.4f}")
print("Polynomial Regression:", f"MAE: {mae_poly:.4f}, MSE: {mse_poly:.4f}, R2:
{r2_poly:.4f}")
print("Support Vector Regression:", f"MAE: {mae_svr:.4f}, MSE: {mse_svr:.4f}, R2:
{r2_svr:.4f}")

# --- Plot Predictions vs Actual Values ---
def plot_results(y_test, y_pred, title):
  plt.figure(figsize=(8, 6))
  plt.scatter(y_test, y_pred, alpha=0.5)
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r', linewidth=2)
  plt.xlabel('Actual Values')
```

```
    plt.ylabel('Predicted Values')
    plt.title(title)
    plt.show()

plot_results(y_test, y_pred_mlr, "Linear Regression: Actual vs Predicted")
plot_results(y_test, y_pred_poly, "Polynomial Regression: Actual vs Predicted")
plot_results(y_test, y_pred_svr, "SVR: Actual vs Predicted")
```

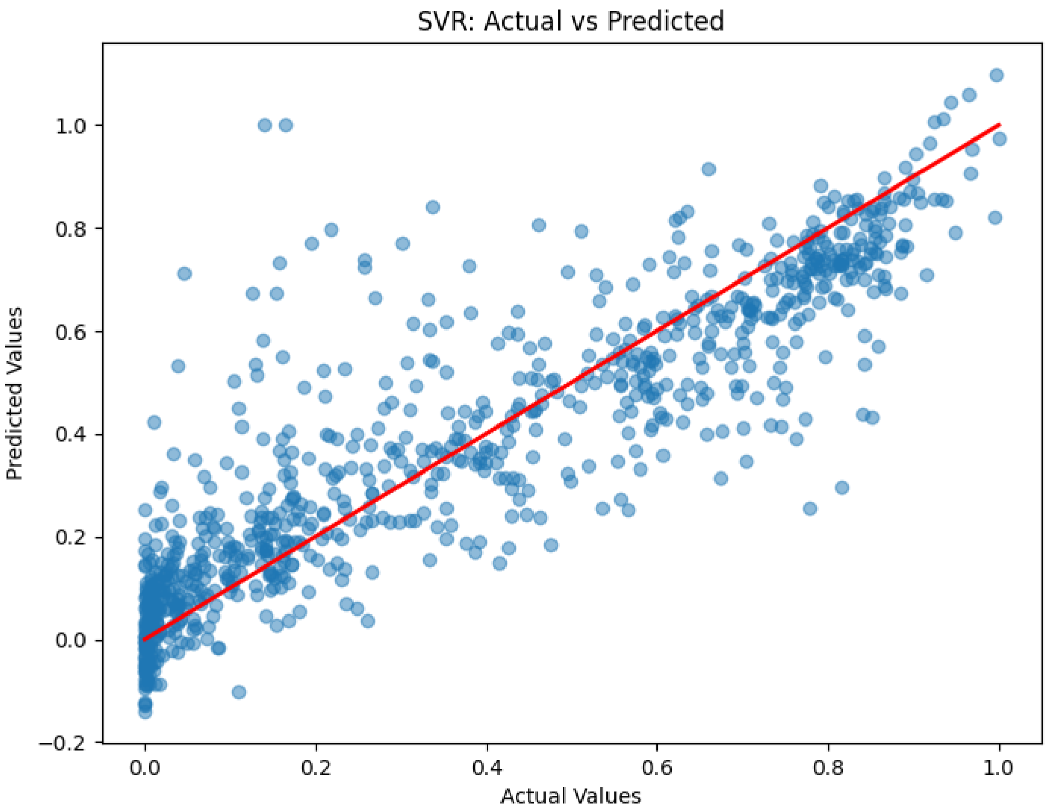
F) The best model is SVR as it has the highest R^2 score and the lowest MAE/MSE.

```
Multiple Linear Regression: MAE: 0.1281, MSE: 0.0276, R2: 0.7177
Polynomial Regression: MAE: 0.1264, MSE: 0.1678, R2: -0.7161
Support Vector Regression: MAE: 0.0997, MSE: 0.0207, R2: 0.7882
```





Polynomial Regression: Actual vs Predicted
Predicted Values
Actual Values
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0



SVR: Actual vs Predicted
Predicted Values
Actual Values
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0