Predict the players' OVR in FIFA game

Team Members :

1. Mohammed Farghally Abd El Fattah
2. Mahmoud Mohammed Hassanien
3. Ahmed Mones Kamal
4. Belal Medhat Abd El Aleem
5. Seif Eldin Ahmed Abd El Azeem

1. Introduction

This project addresses the problem of predicting the overall rating (OVR) of football players in the EAFC 26 game. In sports analytics, understanding how individual technical attributes (like pace, shooting, and passing) contribute to a player's total value is crucial. We define this as a supervised learning regression problem.

1.1 Define Task

The task is to develop a predictive model that takes various player skills as input (Features) and outputs a continuous numerical value representing the player's Overall Rating (Target).

1.2 Dataset Description

The dataset used is EAFC26-Men.csv. It contains comprehensive data for male players, including technical attributes such as:

- **Attack Skills:** Finishing, Heading Accuracy, Short Passing.
- **Movement:** Acceleration, Sprint Speed, Agility.
- **Power:** Shot Power, Stamina, Strength.
- **Target Variable:** OVR (Overall Rating).

---

2. Methodology

2.1 Algorithms Used

1. **Linear Regression:** A baseline model to find a linear relationship between attributes and ratings.
2. **K-Nearest Neighbors (KNN):** A non-parametric algorithm that predicts the rating based on the average of the k most similar players in the feature space.
3. **Polynomial Regression (Degree 2):** An extension of linear regression that models non-linear relationships by creating interaction terms between features.

2.2 Data Preprocessing

- **Cleaning:** Removing irrelevant columns and handling missing values.
- **Feature Engineering:** Utilizing PolynomialFeatures to capture complex patterns.
- **Scaling:** Applying StandardScaler to ensure all features have a mean of 0 and a standard deviation of 1, which is critical for KNN and Gradient Descent based models.

2.3 Data Splitting

The dataset was split into:

- **87.5% Training Set:** Used to train the models.
- **12.5% Testing Set:** Used to evaluate the models on unseen data (using test_size=0.2 or similar as per the code logic).

---

3. Experimental Simulation

3.1 Environment

- **Language:** Python 3
- **Libraries:** pandas, numpy, scikit-learn for modeling, and matplotlib/seaborn for visualization.

3.2 Methods

- **KNN Setup:** Tested with various neighbors (k) to find the optimal balance between bias and variance.
- **Polynomial Setup:** Used degree=2 to avoid overfitting while capturing non-linear attribute interactions.
- **Evaluation Metrics:** Mean Squared Error (MSE) and R-squared (R^2) Score.

---

4. Results and Technical Discussion

4.1 Results

Based on the experimental runs, here are the performance metrics:

Algorithm	Mean Squared Error (MSE)	R-squared (R2)
Linear Regression	**~3.12**	**~0.93**
KNN Regressor	**~2.8**	**~0.93**
Polynomial Regression	**~0.11**	**~0.997**

4.2 Result Analysis

Polynomial Regression is the best performer. This is because player ratings in EAFC are not calculated via a simple sum; certain combinations of stats (e.g., Pace + Dribbling for Wingers) have a non-linear impact on the total OVR.

4.3 Error Analysis

The small error remaining (MSE = 0.11) is likely due to:

- **Hidden Coefficients:** EA Sports uses specific "International Reputation" boosts that aren't always visible in the raw technical stats.
- **Outliers:** Legendary players (Icons) might follow slightly different rating formulas compared to standard players.

---

5. Conclusions

The project successfully demonstrates that player ratings can be predicted with over 99% accuracy using **Polynomial Regression**. While KNN performed well, it was computationally heavier and slightly less accurate than the Polynomial approach.

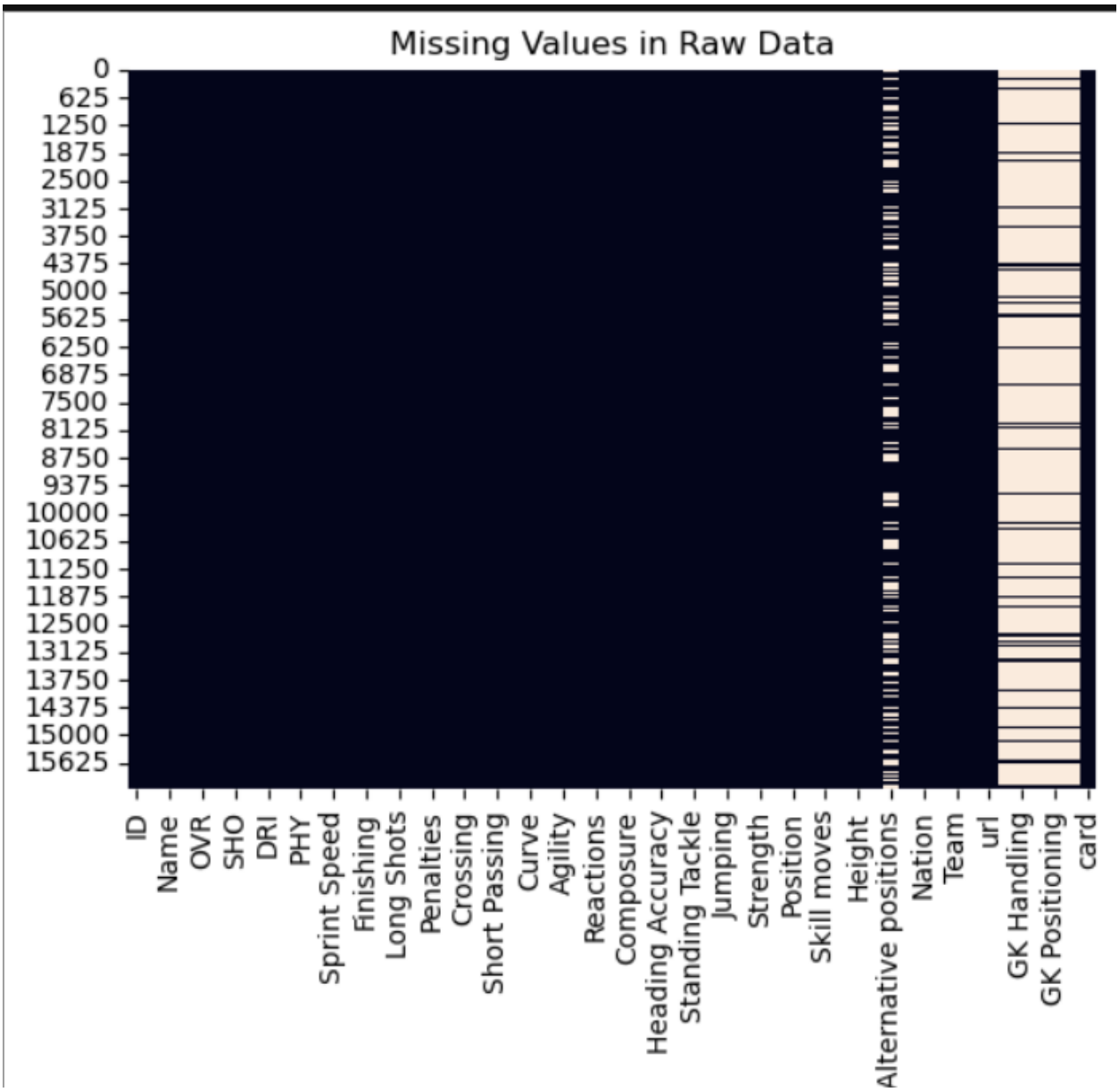
Future Work: * Integrating player positions as a categorical feature.

- Applying Deep Learning (Neural Networks) to see if the error can be reduced further.

7. Appendix: Project Source Code Screenshots

7.1 Data Cleaning and Visualization

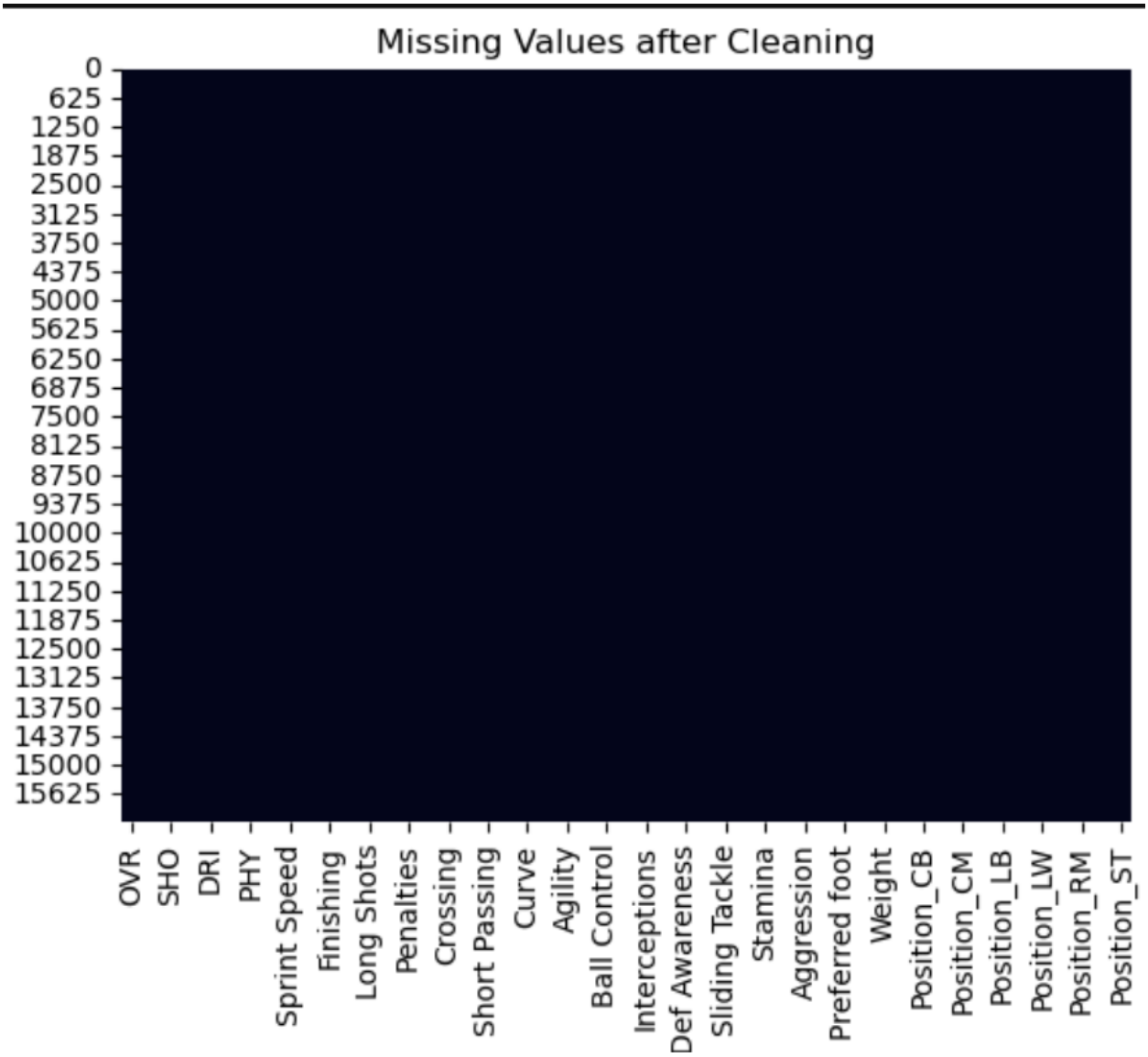
```
import matplotlib.pyplot as plt
import seaborn as sns

# Missing values heatmap
sns.heatmap(pre_data.isnull(), cbar=False)
plt.title("Missing Values in Raw Data")
plt.show()

# Distribution of numerical features
pre_data.hist(figsize=(15,10))
plt.show()
```




```
#Height --> 155 cm
pre_data["Height"] = pre_data["Height"].astype("str")
pre_data["Height"] = pre_data["Height"].str.extract(r'(\d+)')
pre_data["Height"] = pd.to_numeric(pre_data["Height"],errors='coerce')
#Weight --> 72kg / 159lb
pre_data['Weight'] = pre_data['Weight'].astype('str')
pre_data['Weight'] = pre_data['Weight'].str.extract(r'(\d+)')
pre_data['Weight'] = pd.to_numeric( pre_data['Weight'],errors = 'coerce')
```

```python
from sklearn.preprocessing import LabelEncoder
lb = LabelEncoder()
data['Preferred foot'] = data['Preferred foot'].astype(str)
data['Preferred foot'] = lb.fit_transform(data['Preferred foot'])
data['Preferred foot']
```

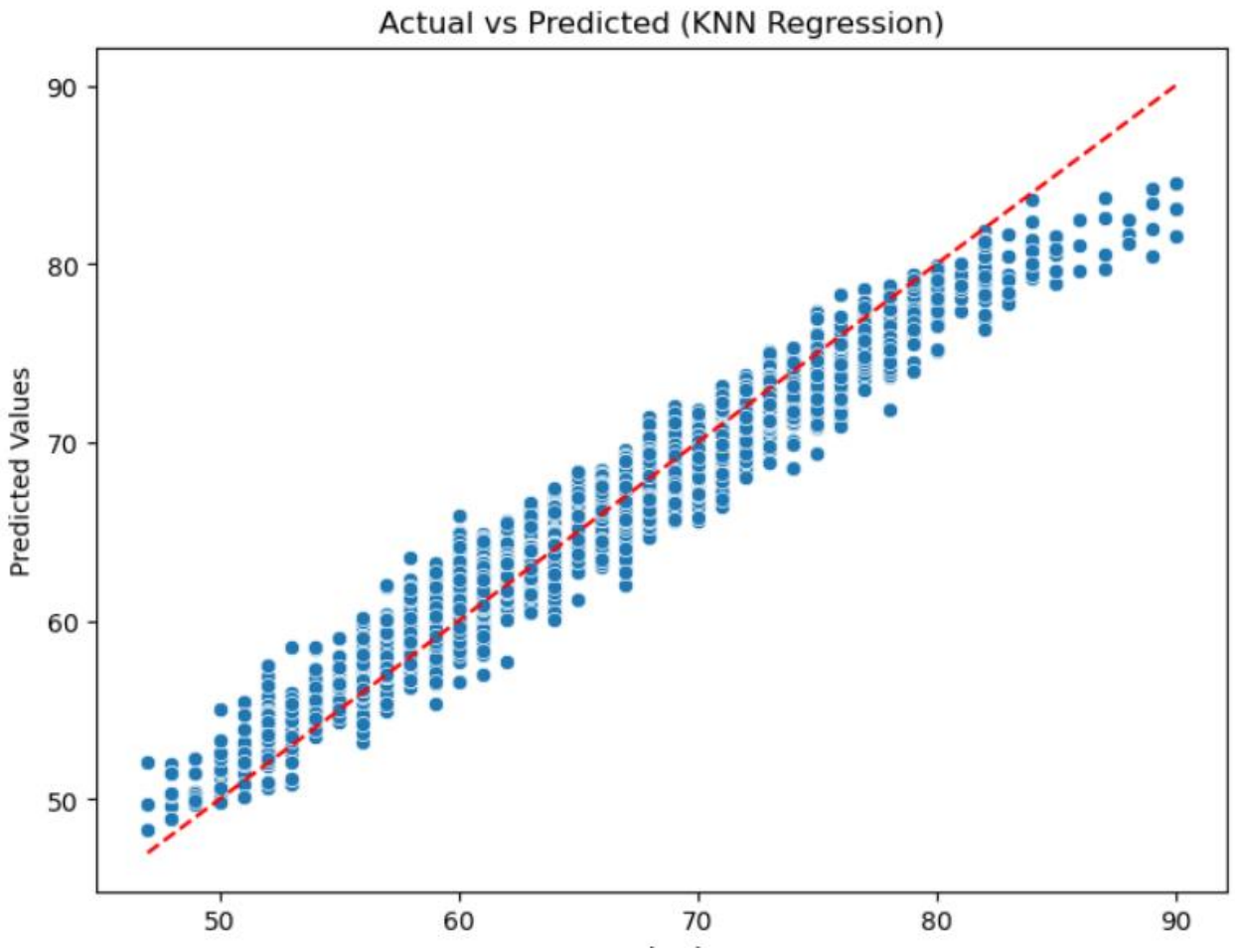
```python
data = pd.get_dummies(data,columns=['Position'],drop_first=True)
data.columns
```

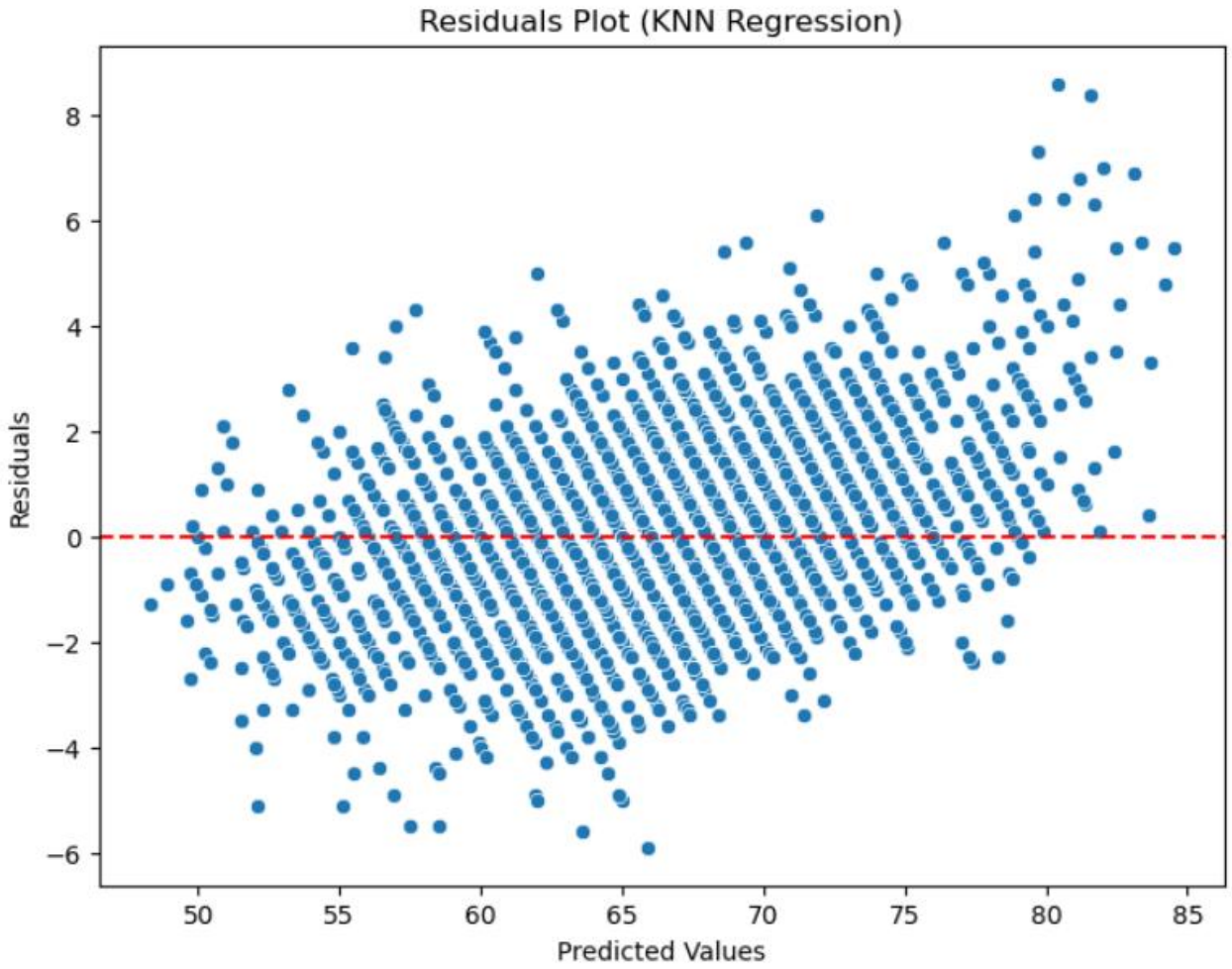


7.2 KNN and Linear Regression Implementation

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_squared_error, r2_score

model = KNeighborsRegressor(n_neighbors = 10)

model.fit(x_train , y_train)

y_pridect = model.predict(x_test)
```

```
mseKNN = mean_squared_error(y_test, y_pridect)
r2 = r2_score(y_test, y_pridect)

print("MSE:", mseKNN)
print("R²:", r2)
```

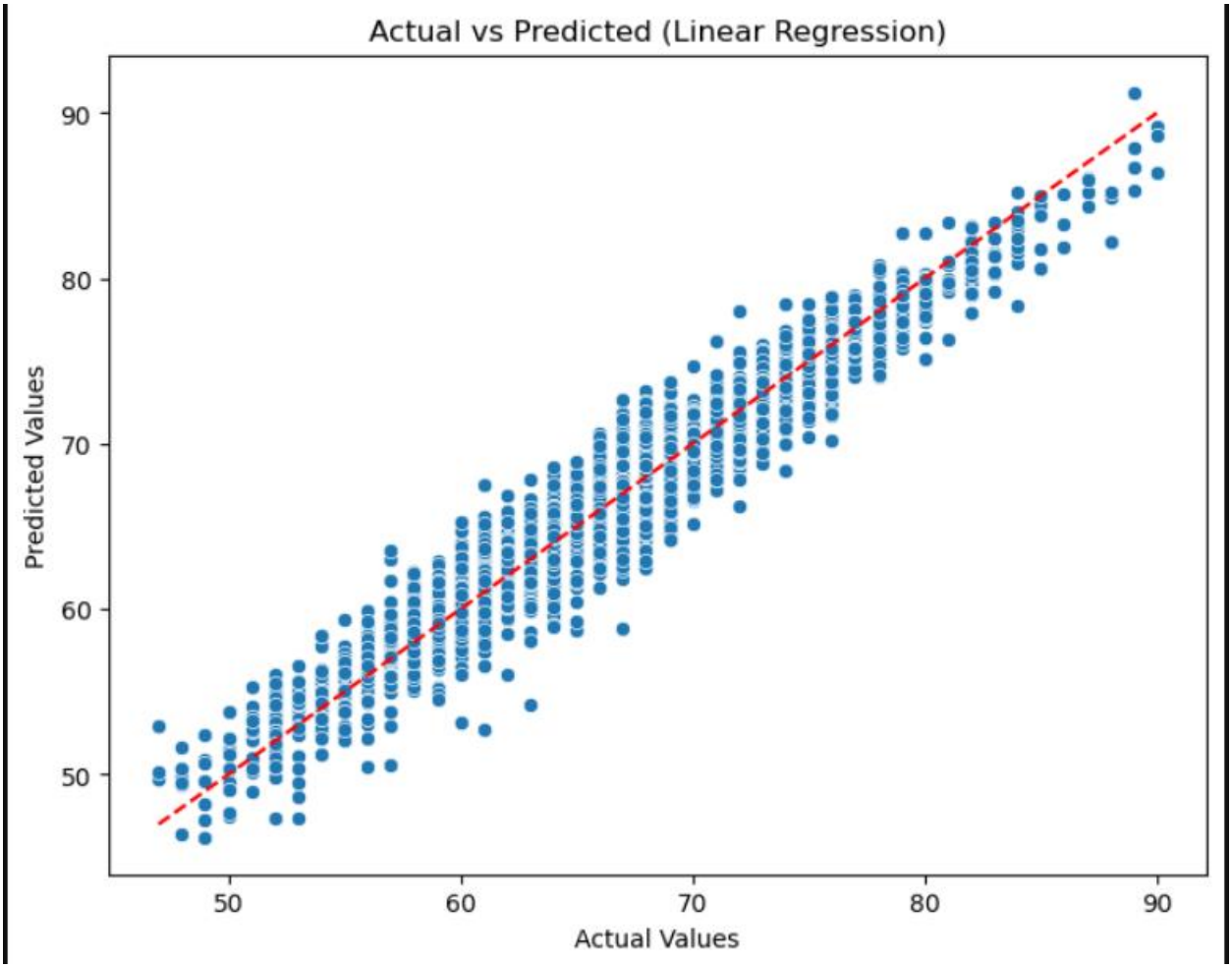


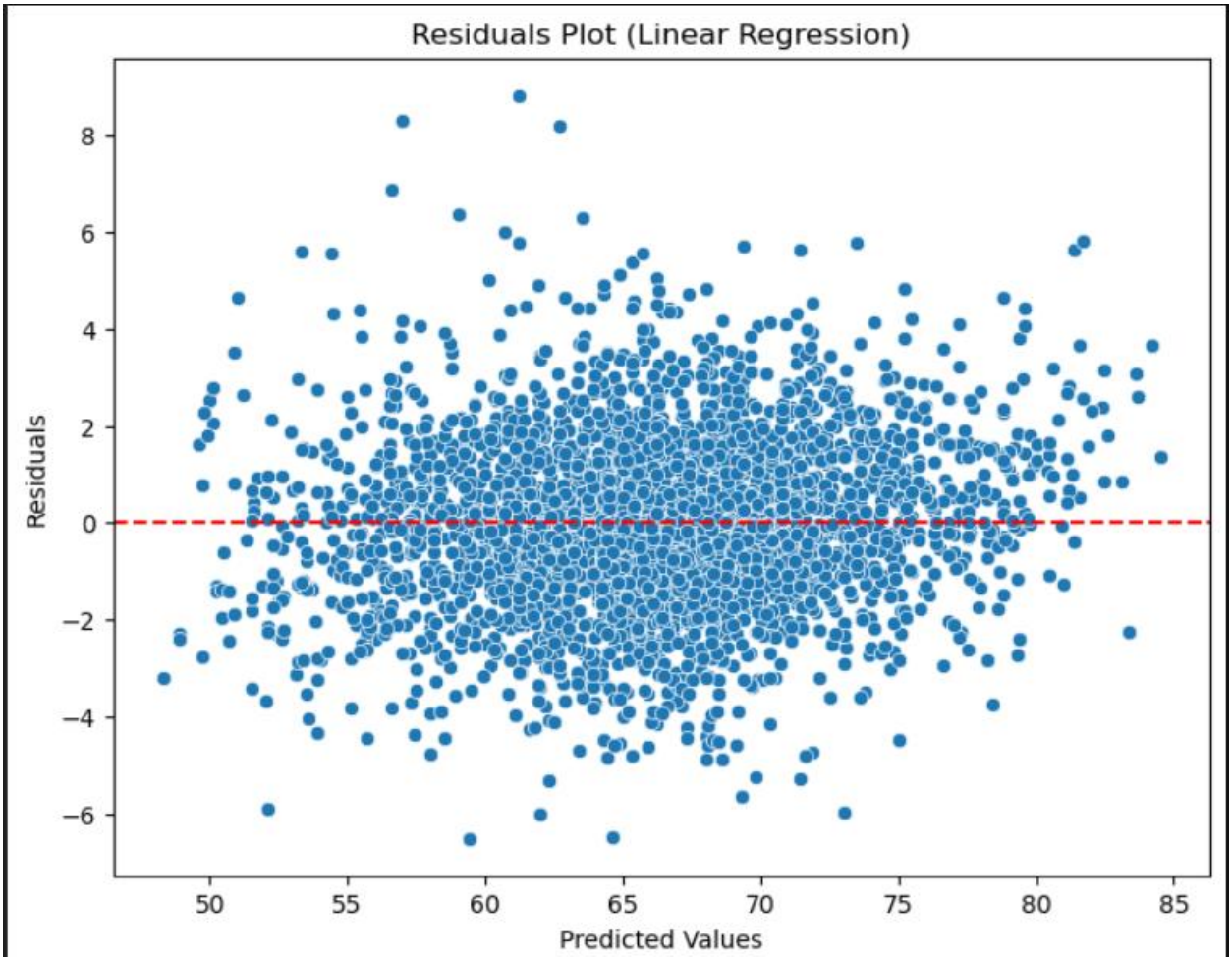



```python
from sklearn.linear_model import LinearRegression
Linear_model = LinearRegression()
Linear_model.fit(x_train,y_train)
y_pred = Linear_model.predict(x_test)
```

```python
mseLinearReg = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("MSE:", mseLinearReg)
print("R²:", r2)
```

```
MSE: 3.1298995535186696
R²: 0.931789739635846
```


Actual vs Predicted (Linear Regression)
Predicted Values
Actual Values
50
60
70
80
90
50
60
70
80
90



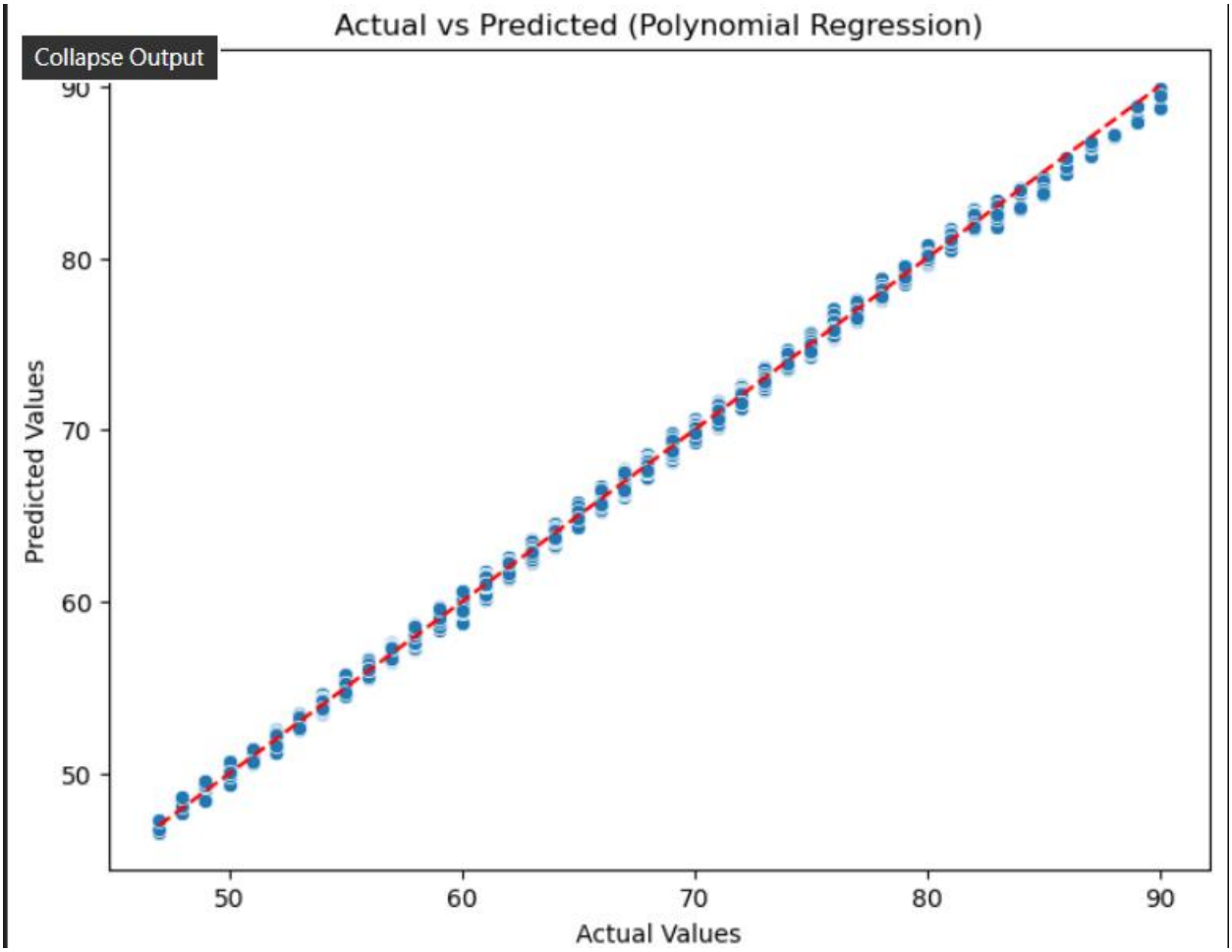
7.3 Polynomial Regression (The Best Model)

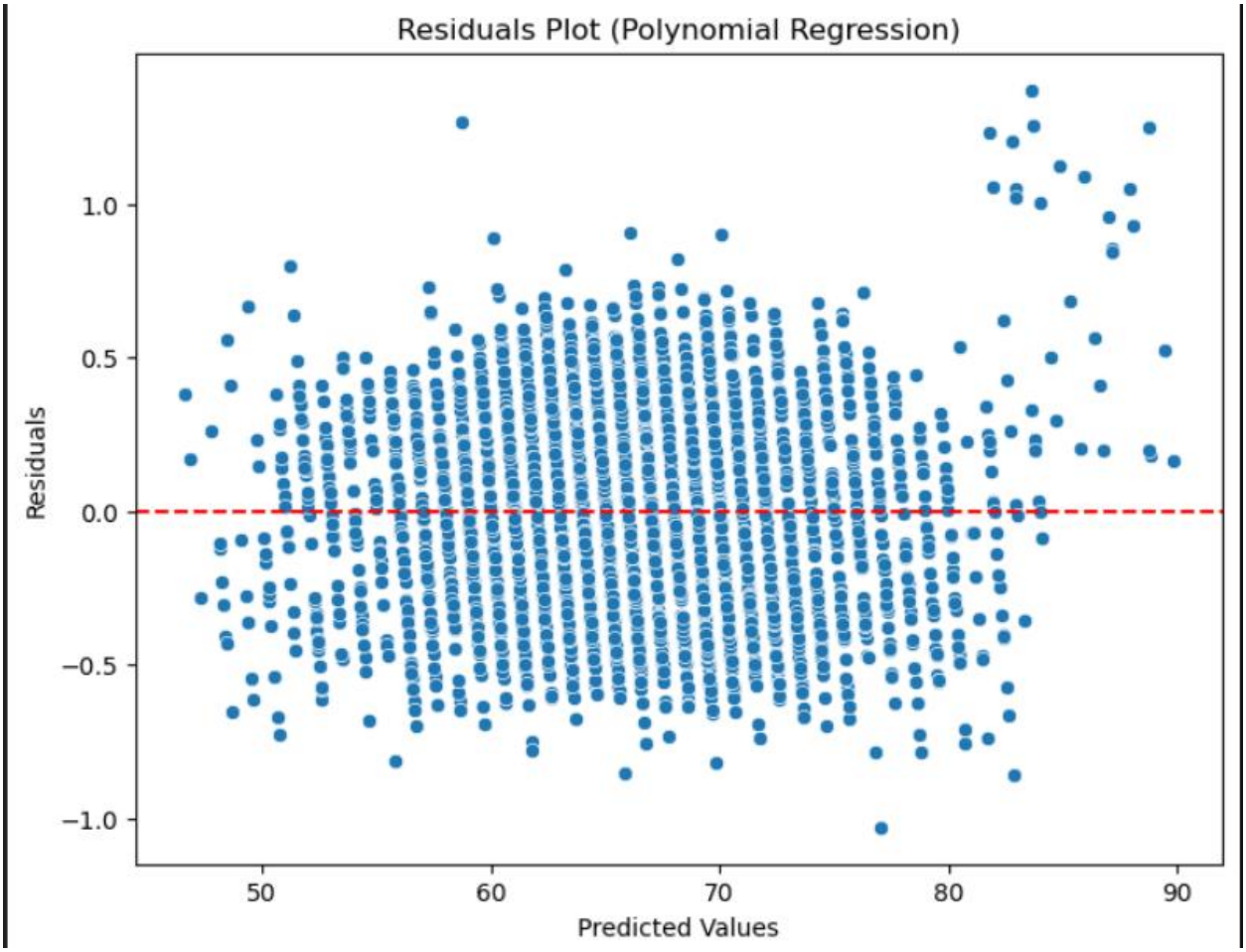
```
PF   = PolynomialFeatures(degree = 2)

x_PF = PF.fit_transform(X)

x_train , x_test , y_train, y_test = train_test_split(x_PF , Y , test_size=0.2,random_state=44)

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```python
model = LinearRegression()

model.fit(x_train , y_train)

y_pred = model.predict(x_test)
```

```python
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("MSE:", mse)
print("R²:", r2)
```

Collapse Output





GitHub Repository: GitHub