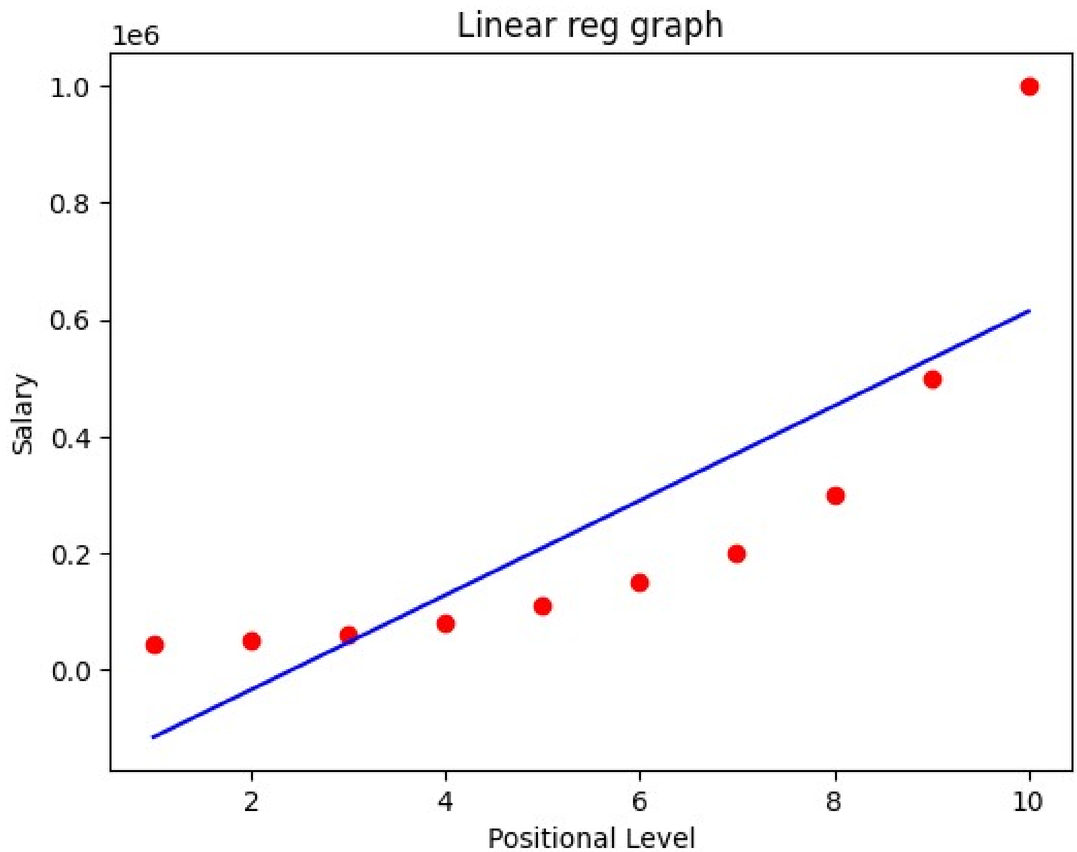
IMPORTING LIBRARIES

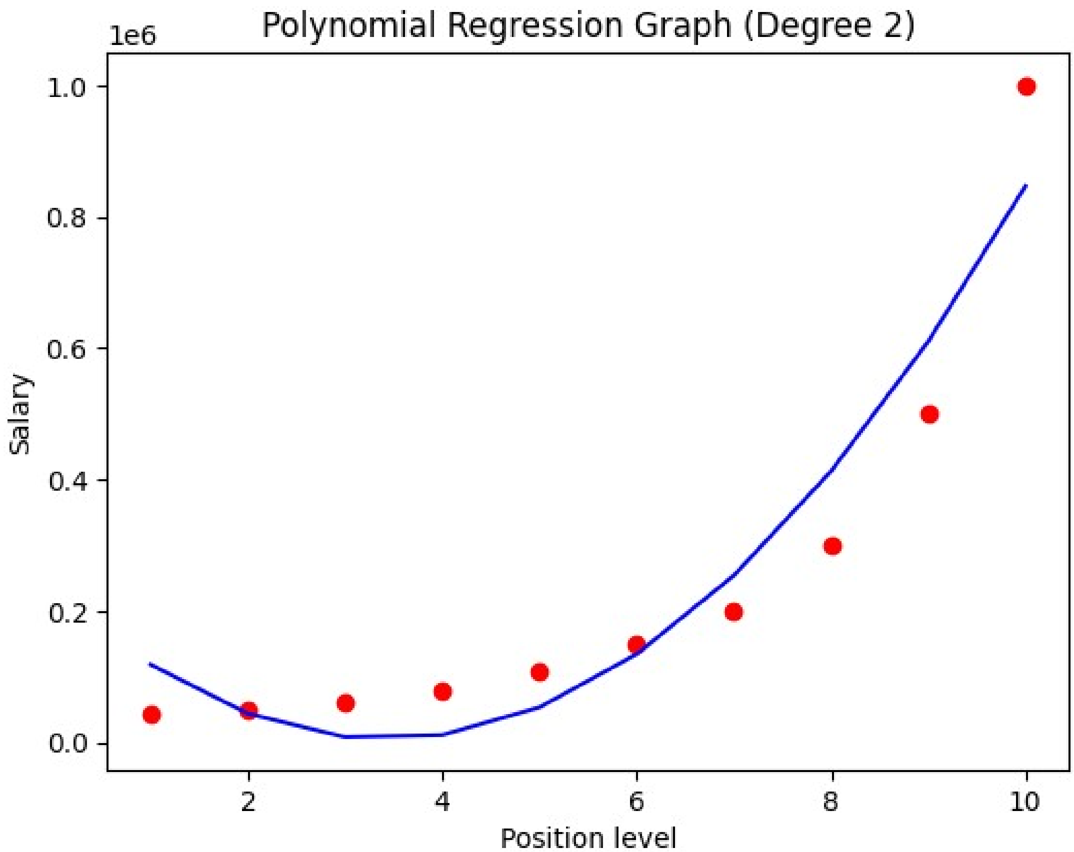
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv(r"D:\DataScienceGenai\Polynomial_regression\
Polynomial regression\Polynomial regression\1.POLYNOMIAL REGRESSION\
emp_sal.csv")
data
```

	Position	Level	Salary
0	Jr Software Engineer	1	45000
1	Sr Software Engineer	2	50000
2	Team Lead	3	60000
3	Manager	4	80000
4	Sr manager	5	110000
5	Region Manager	6	150000
6	AVP	7	200000
7	VP	8	300000
8	CTO	9	500000
9	CEO	10	1000000

```
X = data.iloc[:,1:2].values
y = data.iloc[:,2].values

from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X,y)

LinearRegression()

plt.scatter(X,y, color = 'red')
plt.plot(X,lin_reg.predict(X),color = 'blue')
plt.title("Linear reg graph")
plt.xlabel("Positional Level")
plt.ylabel("Salary")
plt.show()
```




```
lin_model_pred = lin_reg.predict(([[6.5]]))
print(f"Linear Regression Prediction for 6.5: {lin_model_pred}")

Linear Regression Prediction for 6.5: [330378.78787879]

from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=2)
X_poly = poly_reg.fit_transform(X)

# Create and train the Polynomial Regression model
poly_regressor = LinearRegression()
poly_regressor.fit(X_poly, y)

LinearRegression()

plt.scatter(X,y, color = 'red')
plt.plot(X,poly_regressor.predict(X_poly), color = 'blue') # Use
X_poly for prediction
plt.title("Polynomial Regression Graph (Degree 2)")
plt.xlabel("Position level")
plt.ylabel("Salary")
plt.show()
```



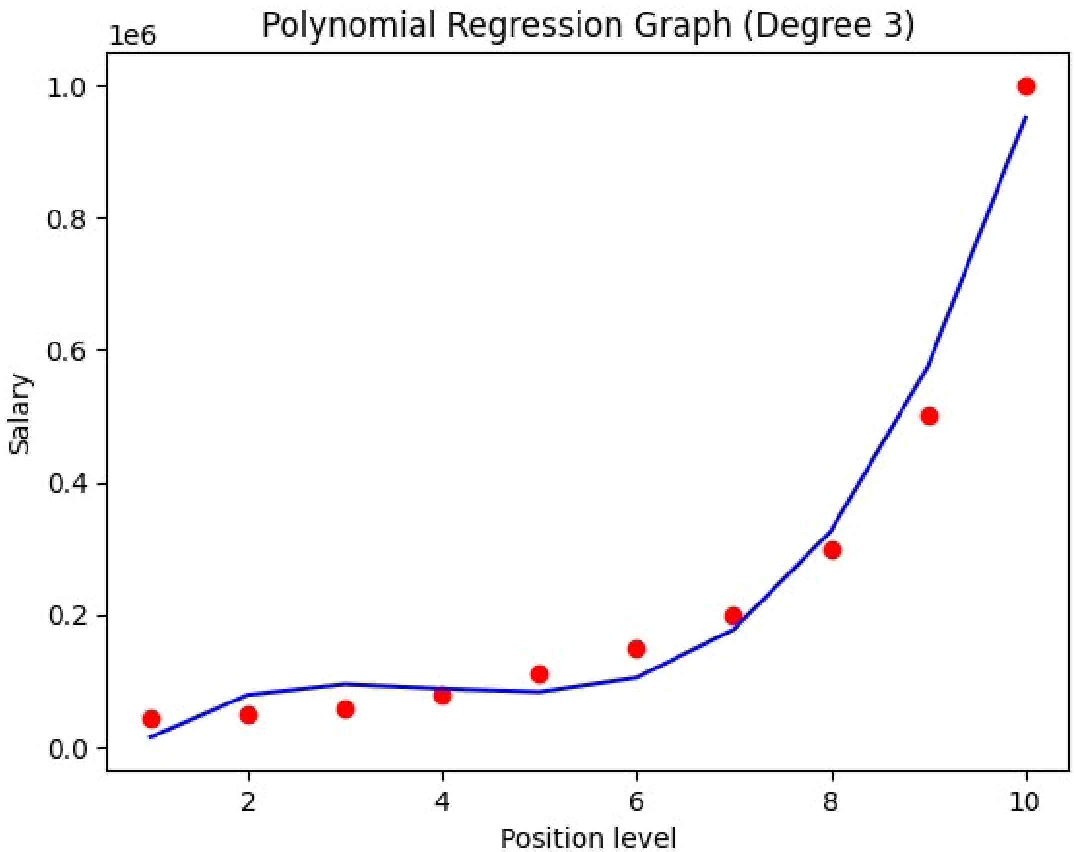

```
poly_model_pred = poly_regressor.predict(poly_reg.transform([[6.5]]))
# Use poly_reg.transform
print(f"Polynomial Regression Prediction for 6.5: {poly_model_pred}")
```

```
Polynomial Regression Prediction for 6.5: [189498.10606061]
```

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=3)
X_poly = poly_reg.fit_transform(X)

# Create and train the Polynomial Regression model
poly_regressor = LinearRegression()
poly_regressor.fit(X_poly, y)
```

```
LinearRegression()
```

```
plt.scatter(X,y, color = 'red')
plt.plot(X,poly_regressor.predict(X_poly), color = 'blue') # Use
X_poly for prediction
plt.title("Polynomial Regression Graph (Degree 3)")
plt.xlabel("Position level")
plt.ylabel("Salary")
plt.show()
```




```
poly_model_pred = poly_regressor.predict(poly_reg.transform([[6.5]]))
# Use poly_reg.transform
print(f"Polynomial Regression Prediction for 6.5: {poly_model_pred}")
```

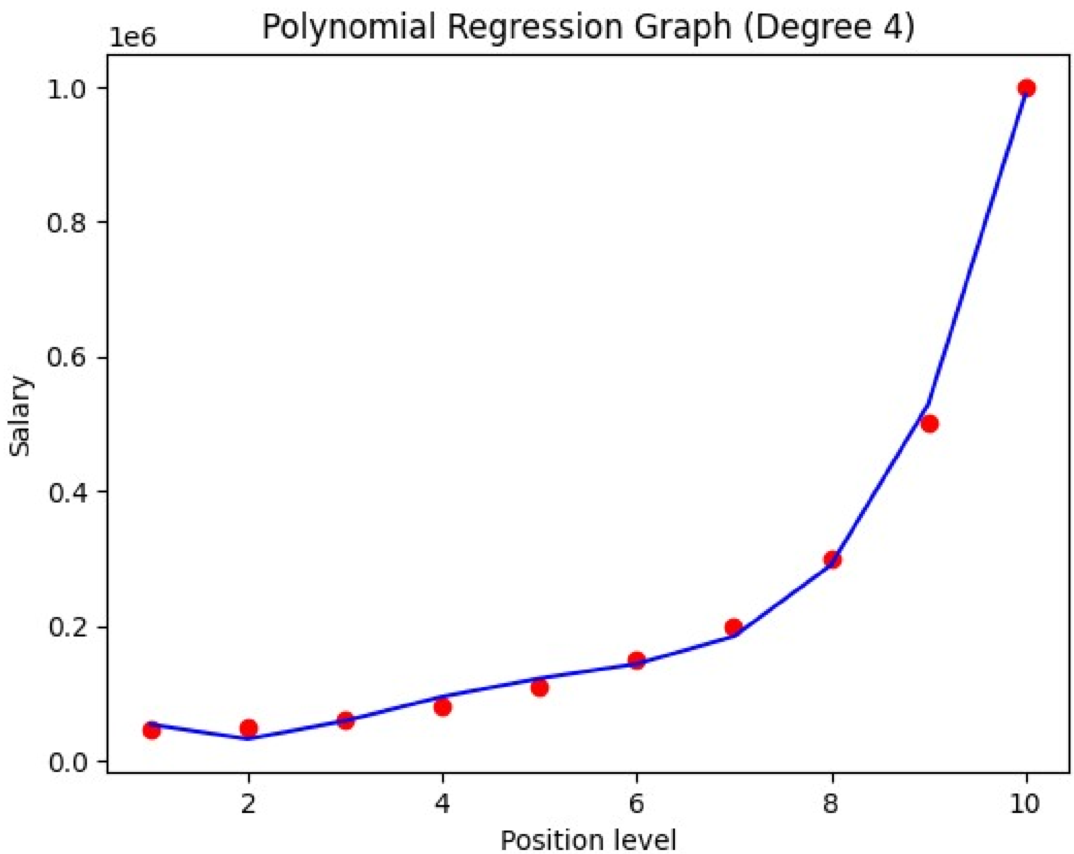
```
Polynomial Regression Prediction for 6.5: [133259.46969697]
```

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=4)
X_poly = poly_reg.fit_transform(X)

# Create and train the Polynomial Regression model
poly_regressor = LinearRegression()
poly_regressor.fit(X_poly, y)
```

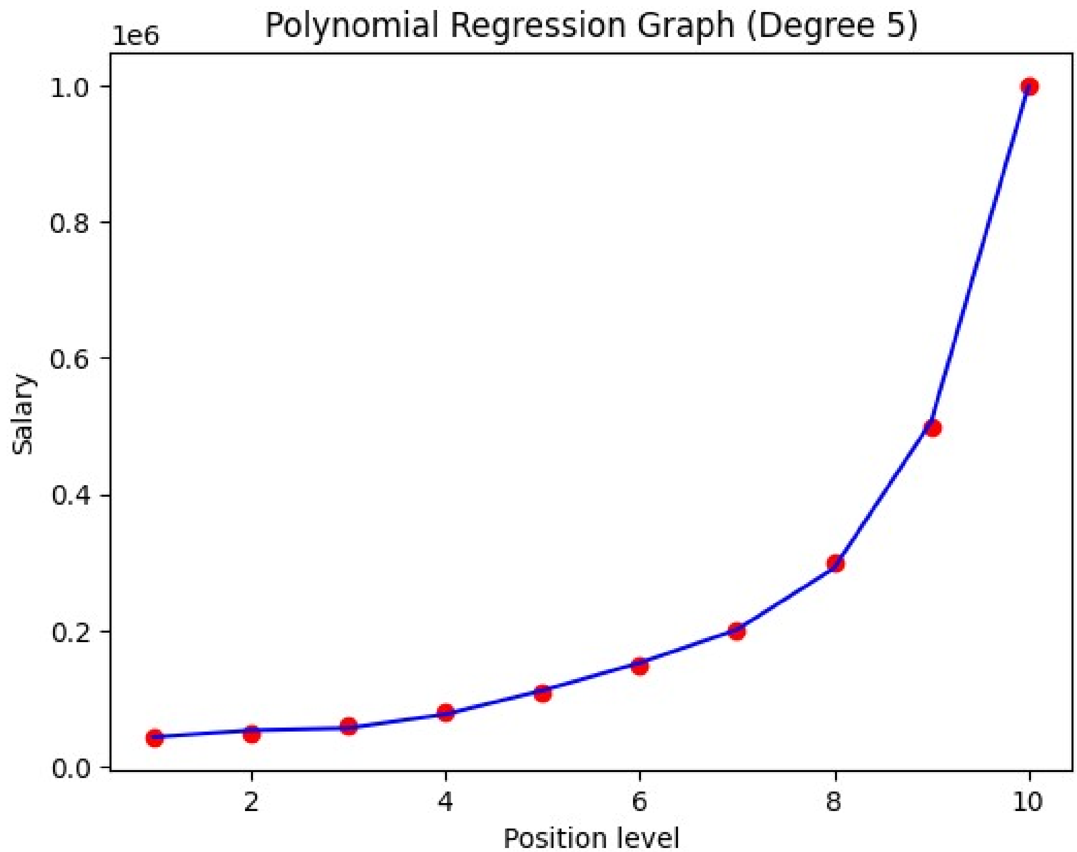
```
LinearRegression()
```

```
plt.scatter(X,y, color = 'red')
plt.plot(X,poly_regressor.predict(X_poly), color = 'blue') # Use
X_poly for prediction
plt.title("Polynomial Regression Graph (Degree 4)")
plt.xlabel("Position level")
plt.ylabel("Salary")
plt.show()
```

```
poly_model_pred = poly_regressor.predict(poly_reg.transform([[6.5]]))
# Use poly_reg.transform
print(f"Polynomial Regression Prediction for 6.5: {poly_model_pred}")
```




```
Polynomial Regression Prediction for 6.5: [158862.45265155]

from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=5)
X_poly = poly_reg.fit_transform(X)

# Create and train the Polynomial Regression model
poly_regressor = LinearRegression()
poly_regressor.fit(X_poly, y)

plt.scatter(X,y, color = 'red')
plt.plot(X,poly_regressor.predict(X_poly), color = 'blue') # Use
X_poly for prediction
plt.title("Polynomial Regression Graph (Degree 5)")
plt.xlabel("Position level")
plt.ylabel("Salary")
plt.show()
```

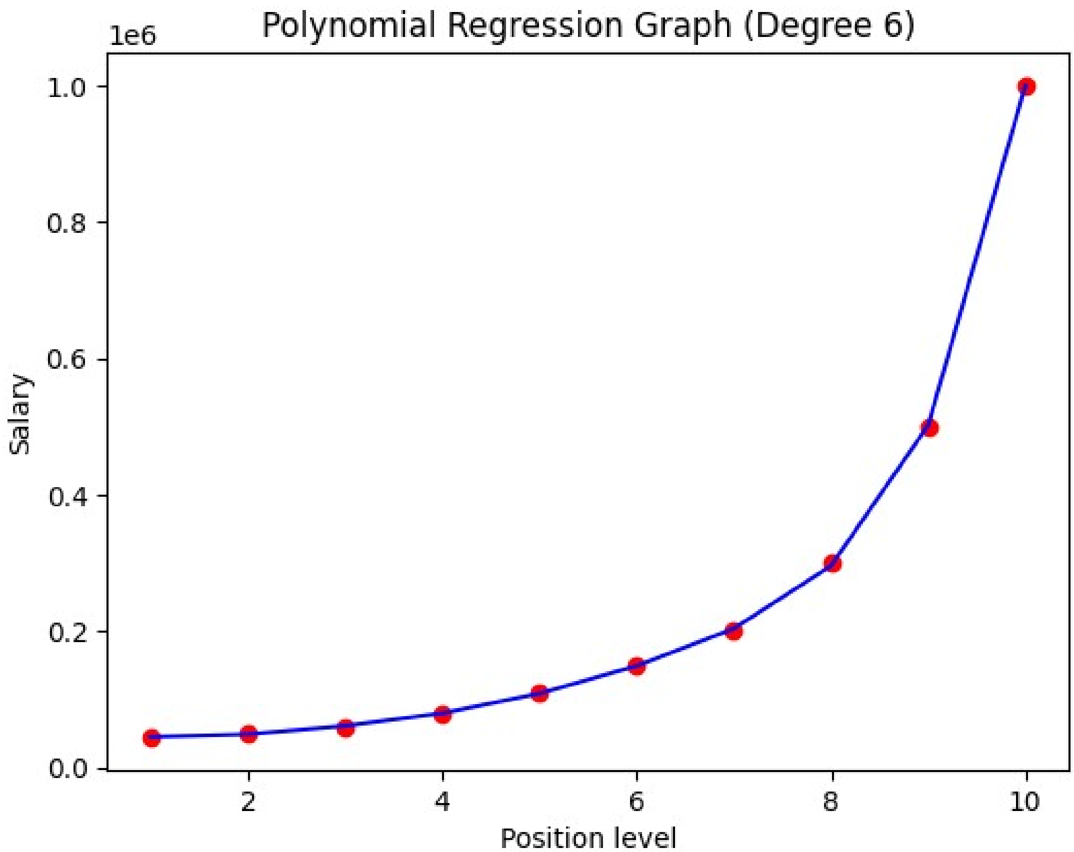



```
poly_model_pred = poly_regressor.predict(poly_reg.transform([[6.5]]))
# Use poly_reg.transform
print(f"Polynomial Regression Prediction for 6.5: {poly_model_pred}")
```

```
Polynomial Regression Prediction for 6.5: [174878.07765173]
```

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=6)
X_poly = poly_reg.fit_transform(X)

# Create and train the Polynomial Regression model
poly_regressor = LinearRegression()
poly_regressor.fit(X_poly, y)

plt.scatter(X,y, color = 'red')
plt.plot(X,poly_regressor.predict(X_poly), color = 'blue') # Use
X_poly for prediction
plt.title("Polynomial Regression Graph (Degree 6)")
plt.xlabel("Position level")
plt.ylabel("Salary")
plt.show()

poly_model_pred = poly_regressor.predict(poly_reg.transform([[6.5]]))
```

```python
# Use poly_reg.transform
print(f"Polynomial Regression Prediction for 6.5: {poly_model_pred}")
```

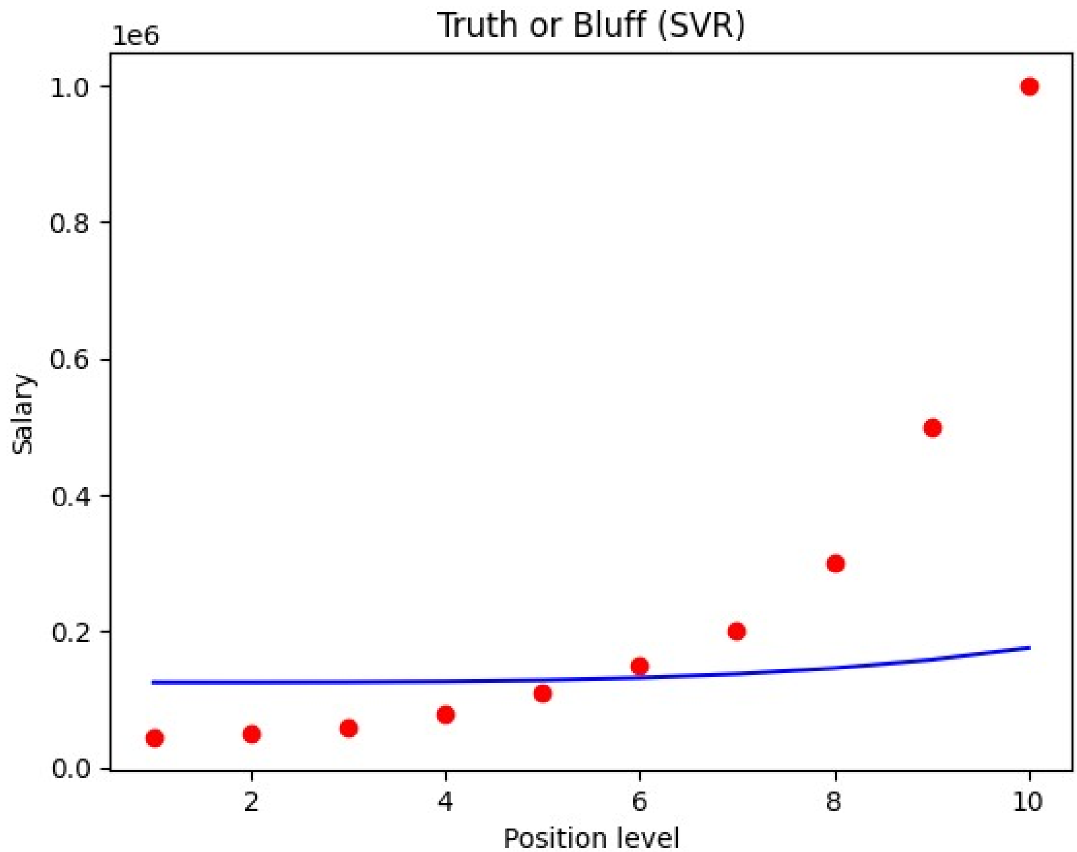



```
Polynomial Regression Prediction for 6.5: [174192.81930603]
```

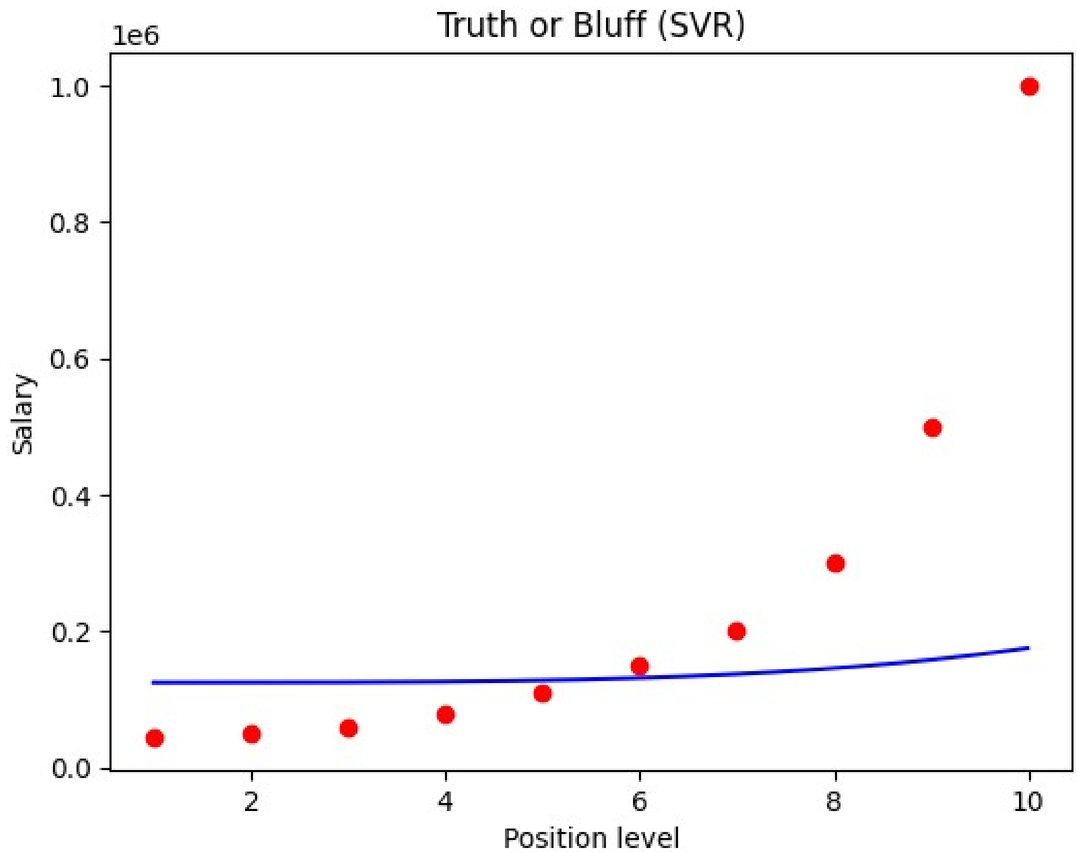
```python
# svm model
from sklearn.svm import SVR
svr_regressor = SVR(kernel='poly',degree = 5,gamma = 'scale' )
svr_regressor.fit(X,y)

svr_model_pred = svr_regressor.predict([[6.5]])
print(svr_model_pred)


# Fitting SVR to the dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'poly',degree = 4)
regressor.fit(X, y)

y_pred_svr = regressor.predict([[6.5]])


# Visualising the SVR results
```

```
plt.scatter(X, y, color = 'red')
plt.plot(X, regressor.predict(X), color = 'blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()


# Visualising the SVR results (for higher resolution and smoother curve)
X_grid = np.arange(min(X), max(X), 0.01) # choice of 0.01 instead of 0.1 step because the data is feature scaled
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

```
[164079.01344549]
```




```
C:\Users\mohap\AppData\Local\Temp\ipykernel_10340\3788301405.py:28:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  X_grid = np.arange(min(X), max(X), 0.01) # choice of 0.01 instead of
0.1 step because the data is feature scaled
```




```
# knn model
from sklearn.neighbors import KNeighborsRegressor
knn_reg_model = KNeighborsRegressor(n_neighbors=4, weights='distance',
p=1, metric='minkowski')
knn_reg_model.fit(X,y)

knn_reg_pred = knn_reg_model.predict([[6.5]])
print(knn_reg_pred)
1
```

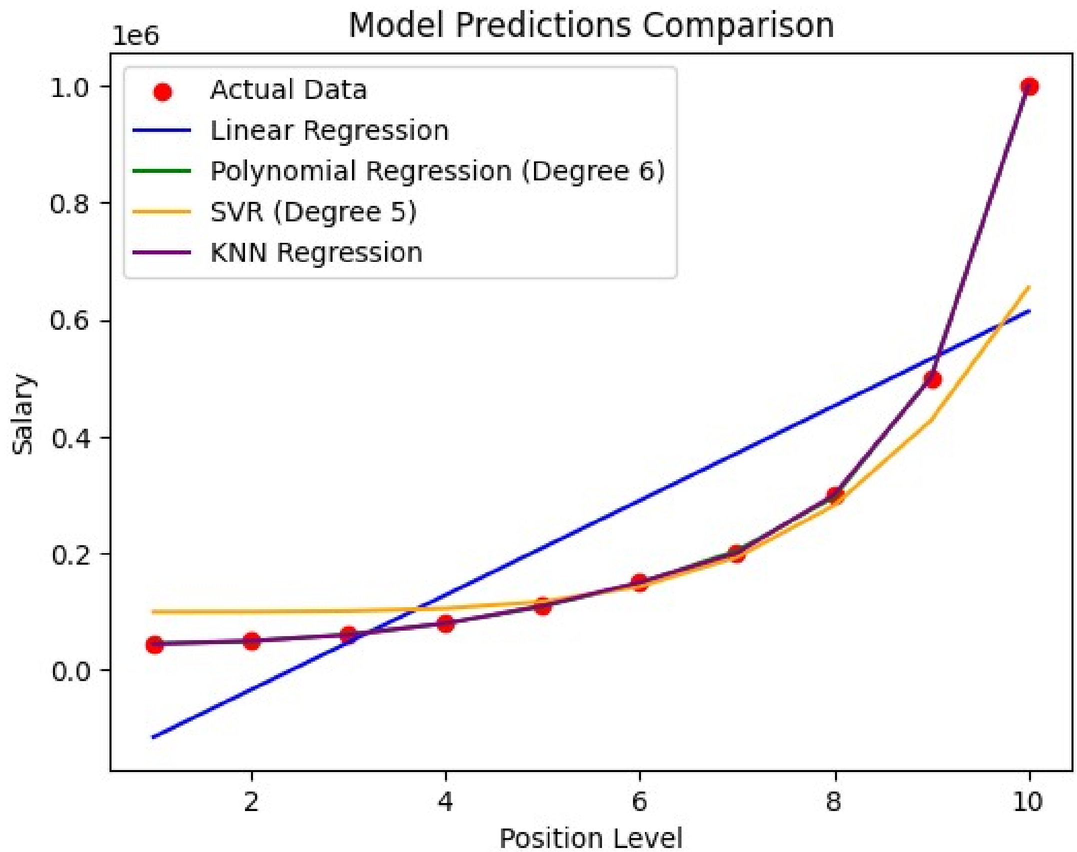
```
[182500.]
```

```
1
```

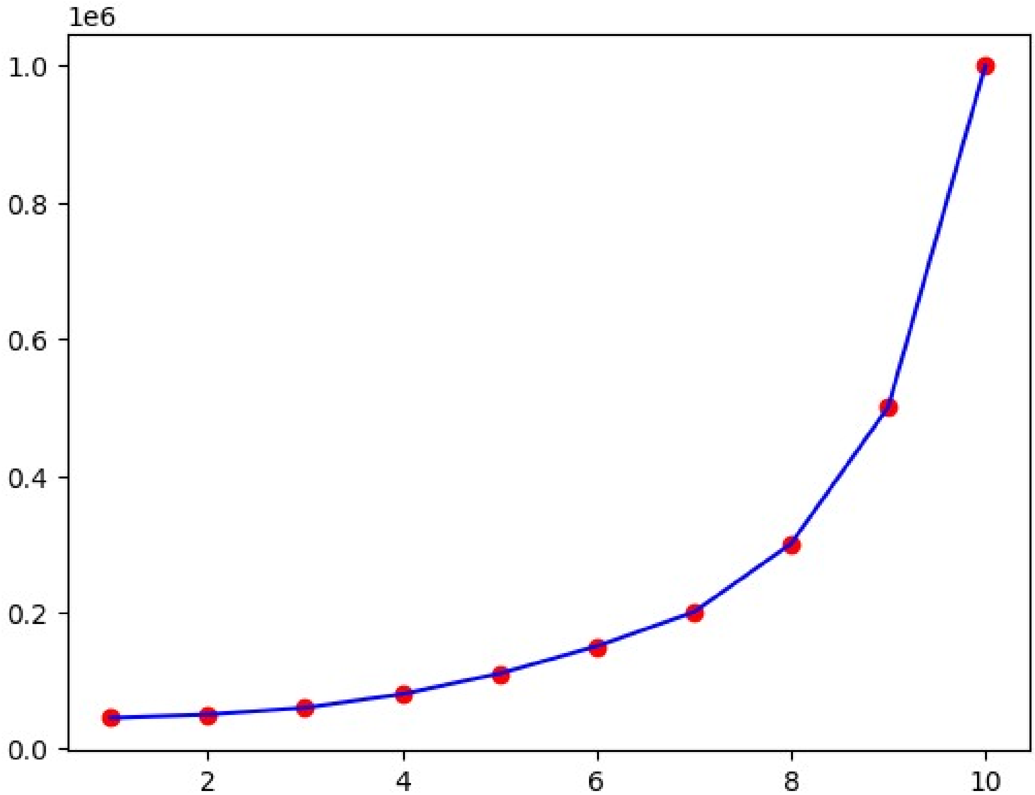
```
# knn model
from sklearn.neighbors import KNeighborsRegressor
knn_reg_model = KNeighborsRegressor(n_neighbors=5, weights='distance',
p=2)
knn_reg_model.fit(X,y)

knn_reg_pred = knn_reg_model.predict([[6.5]])
print(knn_reg_pred)
```

```
[175348.8372093]
```

```python
# Plotting the actual data points
plt.scatter(X, y, color='red', label='Actual Data')

# Plotting Linear Regression predictions
plt.plot(X, lin_reg.predict(X), color='blue', label='Linear
Regression')

# Plotting Polynomial Regression predictions
plt.plot(X, poly_regressor.predict(X_poly), color='green',
label='Polynomial Regression (Degree 6)')

# Plotting SVR predictions
plt.plot(X, svr_regressor.predict(X), color='orange', label='SVR
(Degree 5)')

# Plotting KNN predictions
plt.plot(X, knn_reg_model.predict(X), color='purple', label='KNN
Regression')

# Adding labels and title
plt.title('Model Predictions Comparison')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.legend()
plt.show()
```




```
from sklearn.tree import DecisionTreeRegressor
regrssor_dtr =
DecisionTreeRegressor(criterion='absolute_error',splitter='random',ran
dom_state=0)
regrssor_dtr.fit(X, y)
# Create and train the Decision Tree Regressor

# decision tree

DecisionTreeRegressor(criterion='absolute_error', random_state=0,
                      splitter='random')

y_pred_dtr = regrssor_dtr.predict([[6.5]])
print(y_pred_dtr)

[200000.]

y_pred_svr = regrssor_dtr.predict([[6.5]])
# Visualising the Decision Tree results
plt.scatter(X, y, color = 'red')
plt.plot(X, regrssor_dtr.predict(X), color = 'blue')

[<matplotlib.lines.Line2D at 0x213410bacf0>]
```




```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=regrssor_dtr,
param_grid=param_grid, cv=5, scoring='neg_mean_squared_error',
n_jobs=-1)
grid_search.fit(X, y)

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", -grid_search.best_score_)

Best Parameters: {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 2}
Best Score: 56692500000.0

# RANDOM FOREST REGRESSOR
from sklearn.ensemble import RandomForestRegressor
```

```
rf_regressor = RandomForestRegressor(n_estimators=100,random_state=0,
max_depth=5, min_samples_split=2, min_samples_leaf=1)
# Create and train the Random Forest Regressor
rf_regressor.fit(X, y)
```

```
RandomForestRegressor(max_depth=5, random_state=0)
```

```
# Predicting a new result with Random Forest Regression
y_pred_rf = rf_regressor.predict([[6.5]])
print(y_pred_rf)
```

```
[158300.]
```

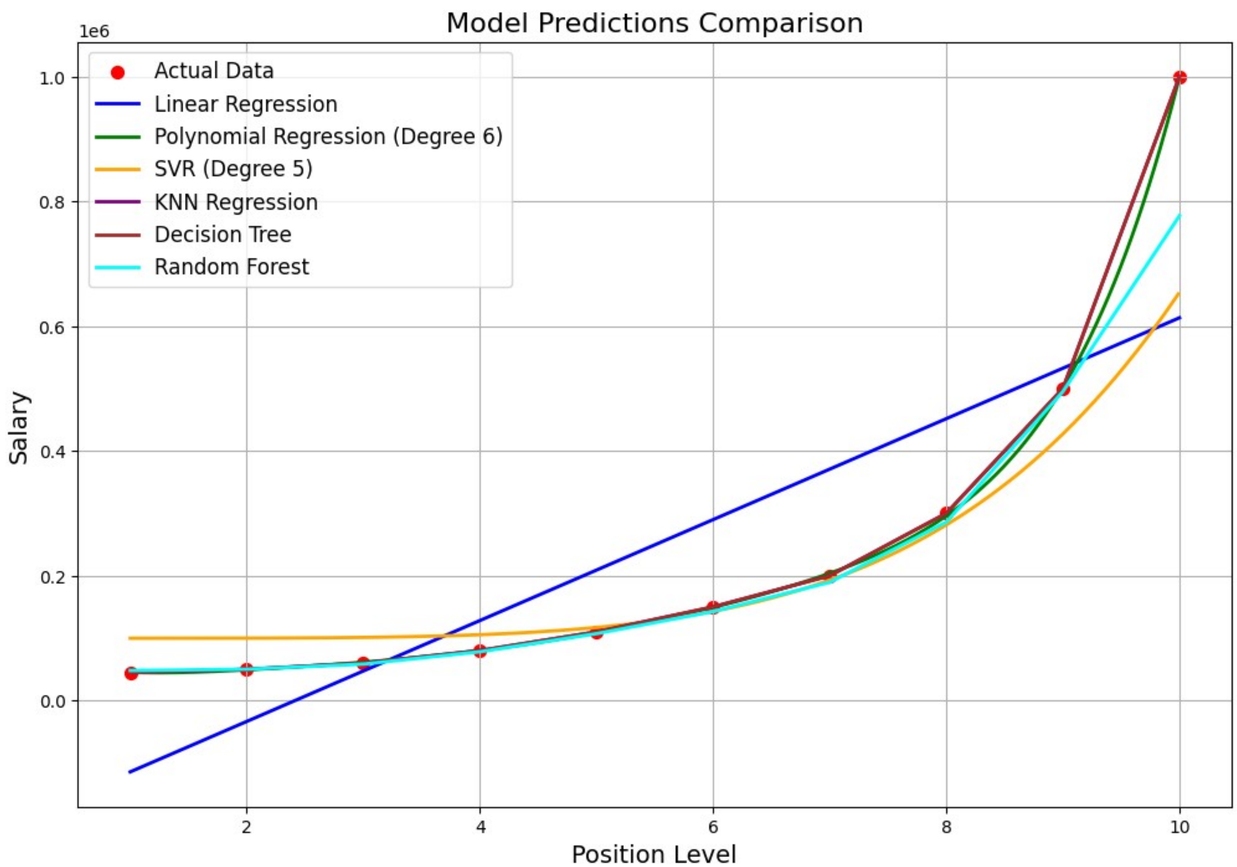
The time taken by all models mentioned above is as follows:

- **Linear Regression Time**: 0.002021 seconds
- **Polynomial Regression Time**: 0.004050 seconds
- **SVR Time**: 0.002304 seconds
- **KNN Time**: 0.005392 seconds
- **Decision Tree Time**: 0.001779 seconds
- **Random Forest Time**: 0.010905 seconds

```
# Actual value for level 6.5 (interpolated from the dataset)
actual_value = 150000

# Predictions from different models
predictions = {
    "Linear Regression": lin_model_pred[0],
    "Polynomial Regression (Degree 6)": poly_model_pred[0],
    "SVR (Degree 5)": svr_model_pred[0],
    "KNN Regression": knn_reg_pred[0],
    "Decision Tree": y_pred_dtr[0],
    "Random Forest": y_pred_rf[0]
}

# Calculate the absolute error for each model
errors = {model: abs(pred - actual_value) for model, pred in
predictions.items()}

# Find the model with the minimum error
best_model = min(errors, key=errors.get)

# Print the best model and its prediction
print(f"Best Model: {best_model}")
print(f"Prediction: {predictions[best_model]}")
print(f"Actual Value: {actual_value}")
```

```
Best Model: Random Forest
Prediction: 158300.0
Actual Value: 150000
```

```python
# Plotting the actual data points
plt.figure(figsize=(12, 8))
plt.scatter(X, y, color='red', label='Actual Data', s=50)

# Plotting Linear Regression predictions
plt.plot(X, lin_reg.predict(X), color='blue', label='Linear 
Regression', linewidth=2)

# Plotting Polynomial Regression predictions
plt.plot(X_grid, poly_regressor.predict(poly_reg.transform(X_grid)), 
color='green', label='Polynomial Regression (Degree 6)', linewidth=2)

# Plotting SVR predictions
plt.plot(X_grid, svr_regressor.predict(X_grid), color='orange', 
label='SVR (Degree 5)', linewidth=2)

# Plotting KNN predictions
plt.plot(X, knn_reg_model.predict(X), color='purple', label='KNN 
Regression', linewidth=2)

# Plotting Decision Tree predictions
plt.plot(X, regrssor_dtr.predict(X), color='brown', label='Decision 
Tree', linewidth=2)

# Plotting Random Forest predictions
plt.plot(X, rf_regressor.predict(X), color='cyan', label='Random 
Forest', linewidth=2)

# Adding labels, title, and legend
plt.title('Model Predictions Comparison', fontsize=16)
plt.xlabel('Position Level', fontsize=14)
plt.ylabel('Salary', fontsize=14)
plt.legend(fontsize=12)
plt.grid(True)
plt.show()
```




```
from sklearn.model_selection import GridSearchCV

# Define parameter grids for each model
param_grids = {
    "Linear Regression": {},  # No hyperparameters to tune for Linear
Regression
    "Polynomial Regression": {"degree": [2, 3, 4, 5, 6]},  #
Polynomial degrees
    "SVR": {
        "kernel": ["poly", "rbf"],
        "degree": [3, 4, 5],
        "C": [1, 10, 100],
        "gamma": ["scale", "auto"]
    },
    "KNN": {
        "n_neighbors": [3, 4, 5, 6],
        "weights": ["uniform", "distance"],
        "p": [1, 2]
    },
    "Decision Tree": {
        "max_depth": [None, 5, 10, 15],
        "min_samples_split": [2, 5, 10],
        "min_samples_leaf": [1, 2, 4]
```

```
    },
    "Random Forest": {
        "n_estimators": [50, 100, 200],
        "max_depth": [None, 5, 10, 15],
        "min_samples_split": [2, 5, 10],
        "min_samples_leaf": [1, 2, 4]
    }
}

# Initialize models
models = {
    "Linear Regression": lin_reg,
    "Polynomial Regression": poly_regressor,
    "SVR": svr_regressor,
    "KNN": knn_reg_model,
    "Decision Tree": regrssor_dtr,
    "Random Forest": rf_regressor
}

# Perform GridSearchCV for each model
best_params = {}
for model_name, model in models.items():
    if model_name == "Polynomial Regression":
        # Special handling for Polynomial Regression
        for degree in param_grids[model_name]["degree"]:
            poly_reg = PolynomialFeatures(degree=degree)
            X_poly = poly_reg.fit_transform(X)
            model.fit(X_poly, y)
            score = model.score(X_poly, y)
            best_params[model_name] = {"degree": degree, "score":
score}
    else:
        grid_search = GridSearchCV(estimator=model,
param_grid=param_grids[model_name], cv=5,
scoring="neg_mean_squared_error", n_jobs=-1)
        grid_search.fit(X, y)
        best_params[model_name] = {"params": grid_search.best_params_,
"score": -grid_search.best_score_}

# Print the best parameters and scores for each model
for model_name, params in best_params.items():
    print(f"{model_name}: Best Parameters: {params.get('params',
params)} | Best Score: {params['score']}")
```

```
Linear Regression: Best Parameters: {} | Best Score: 86661778604.29478
Polynomial Regression: Best Parameters: {'degree': 6, 'score':
0.9999494749253776} | Best Score: 0.9999494749253776
SVR: Best Parameters: {'C': 10, 'degree': 5, 'gamma': 'scale',
'kernel': 'poly'} | Best Score: 6215339280.657389
KNN: Best Parameters: {'n_neighbors': 3, 'p': 1, 'weights':
```

```
'distance'} | Best Score: 71212982671.82991
Decision Tree: Best Parameters: {'max_depth': None,
'min_samples_leaf': 1, 'min_samples_split': 2} | Best Score:
56692500000.0
Random Forest: Best Parameters: {'max_depth': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50} |
Best Score: 61545716000.0
```