Regression on Happiness Score

```
#set up a ColumnTransformer with StandardScaler for numerical features and OneHotEncoder for categorical features.
#set up and training a LinearRegression model using scikit-learn, including data preprocessing steps within a Pipel
#implement polynomial regression
#perform hyperparameter tuning for a polynomial regression model
#evaluate the performance of a regression model on test data
#use OneHotEncoder with handle_unknown='ignore' within a preprocessing pipeline to handle unseen categories during 
#set up and execute cross_val_score or GridSearchCV to perform cross-validation
#perform hyperparameter tuning for an SVM model using grid search
#calculate and display performance metrics
#integrate PolynomialFeatures in a pipeline before an SVM model
#Use different kernels such as linear, polynomial, and RBF
```

› Import Library

[ ] ↳ *1 cell hidden*

⌄ Import Dataset

```
df = pd.read_csv('world_happiness.csv')
df.sample(10)
```

	Unnamed: 0	country	social_support	freedom	corruption	generosity	gdp_per_cap	life_exp	happiness_
22	23	Mexico	67.0	71.0	87.0	120.0	18000	75.6	
112	113	Bangladesh	126.0	27.0	36.0	107.0	4140	73.7	
114	115	Mali	112.0	110.0	107.0	138.0	2100	62.9	
120	121	Ethiopia	119.0	106.0	53.0	99.0	1900	69.1	
128	129	Comoros	143.0	148.0	81.0	62.0	2480	69.1	
141	142	Central African Republic	155.0	133.0	122.0	113.0	794	52.9	
86	87	Bhutan	68.0	59.0	25.0	13.0	9710	74.7	
132	133	Zimbabwe	110.0	96.0	63.0	141.0	2390	62.0	
45	46	Cyprus	90.0	81.0	115.0	39.0	34500	82.0	
55	56	Honduras	84.0	39.0	79.0	51.0	4630	74.3	

› EDA

[ ] ↳ *3 cells hidden*

› Check for null values and preprocessing

[ ] ↳ *3 cells hidden*

Set up a ColumnTransformer with StandardScaler for numerical features and OneHotEncoder for categorical features.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

ct = ColumnTransformer(
    [('standard', StandardScaler(), numerical_features)]
)

X = ct.fit_transform(df)

print(X)
```

```
[[-1.65892081 -1.59118586 -1.69356661 ...  1.08669987  1.13246036
   1.67384369]
 [-1.61488964 -1.56908606 -1.7180244  ...  1.37452586  1.0192638
   1.65220568]
 [-1.63690523 -1.63538547 -1.59573542 ...  2.25263904  1.24565693
   1.63056768]
 ...
 [ 1.6214013   1.72378469  1.53486268 ... -0.89588457 -1.37201358
  -1.63677158]
 [ 1.70946363  1.23758901  1.19245351 ... -0.94300998 -2.95676545
  -1.65840959]
 [ 1.55535454  1.70168488 -0.29947214 ... -0.89100617 -1.99459467
  -1.6800476 ]]
```

Set up and training a LinearRegression model using scikit-learn, including data preprocessing steps within a Pipeline.

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

lr = LinearRegression()

model = Pipeline([
    ('preprocessor', ct),
    ('linear_regression', lr)
])

X = df[numerical_features]
y = df['happiness_score']

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

model.fit(X_train, y_train)

y_pred = model.predict(X_train)

print(y_pred)
```

```
[ 6.30000000e+01  1.26000000e+02  1.13000000e+02  6.10000000e+01
  1.30000000e+02  1.29000000e+02  1.32000000e+02  2.70000000e+01
  8.40000000e+01  1.08000000e+02  1.40000000e+02  4.90000000e+01
  1.14000000e+02  1.33000000e+02  8.30000000e+01  6.70000000e+01
  3.00000000e+00  1.05000000e+02  1.24000000e+02  3.10000000e+01
  7.30000000e+01  6.00000000e+00  1.21000000e+02  6.60000000e+01
  1.00000000e+00  8.90000000e+01  4.00000000e+00  8.00000000e+00
  9.90000000e+01  1.50000000e+02  6.20000000e+01  3.40000000e+01
  1.03000000e+02  1.19000000e+02  7.10000000e+01  1.20000000e+02
  1.20000000e+01  1.48000000e+02  1.09000000e+02  8.10000000e+01
  5.80000000e+01  6.40000000e+01  1.90000000e+01  1.30000000e+01
```

```
 1.47000000e+02  1.42000000e+02  3.50000000e+01  2.80000000e+01
 1.52000000e+02  1.38000000e+02  1.15000000e+02  7.70000000e+01
 2.20000000e+01  1.49000000e+02  4.20000000e+01  1.53000000e+02
 2.00000000e+01  9.80000000e+01  1.02000000e+02  3.90000000e+01
 1.06000000e+02  3.80000000e+01  9.00000000e+01  2.00000000e+00
 6.80000000e+01  7.00000000e+00  5.50000000e+01  1.12000000e+02
 9.40000000e+01  5.60000000e+01  1.04000000e+02  5.90000000e+01
 1.60000000e+01  1.34000000e+02  9.50000000e+01  7.40000000e+01
 1.22000000e+02  5.00000000e+00  9.30000000e+01  8.80000000e+01
 1.40000000e+01  1.16000000e+02  1.25000000e+02  3.60000000e+01
 9.00000000e+00  1.54000000e+02  1.00000000e+02  4.00000000e+01
 4.40000000e+01  2.60000000e+01  6.00000000e+01  7.50000000e+01
 2.10000000e+01 -4.26325641e-14  1.35000000e+02  7.90000000e+01
 3.70000000e+01  1.41000000e+02  5.40000000e+01  4.10000000e+01]
```

Evaluate the performance of a regression model on test data

```
from sklearn.metrics import mean_squared_error

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)

print(f"Mean squared error: {mse}")
```

```
Mean squared error: 4.907815566621513e-28
```

Set up and execute cross_val_score to perform cross-validation

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

scores = cross_val_score(model, X, y, cv=5)

print(f"Cross-validation scores: {scores}")
```

```
Cross-validation scores: [1. 1. 1. 1. 1.]
```

Calculate and display performance metrics

integrate PolynomialFeatures in a pipeline before an SVM model

Use different kernels such as linear, polynomial, and RBF

```
from sklearn.svm import SVR

kernels = ['linear', 'poly', 'rbf']
models = {}

for kernel in kernels:
    model = SVR(kernel=kernel)
    model.fit(X_train, y_train)

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

    train_mse = mean_squared_error(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)

    models[kernel] = model
```

```
        print(f"\nKernel: {kernel}")
        print("Training accuracy:", 100-train_mse)
        print("Testing accuracy:", 100-test_mse)
```