StageZone.sql

```
-- -----------------------------------------------------
-- Table Hapiness
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS  Hapiness (
  `hapiness_id` INT NOT NULL AUTO_INCREMENT,
  `country_name` VARCHAR(45) NULL,
  `year` INT NULL,
  `life_ladder` DECIMAL(10,3) NULL,
  `gdp_per_capita` DECIMAL(10,3) NULL,
  `positive_affect` DECIMAL(10,3) NULL,
  `social_support` DECIMAL(10,3) NULL,
  `life_expancy` DECIMAL(10,3) NULL,
  `freedom_choice` DECIMAL(10,3) NULL,
  `generocity` DECIMAL(10,3) NULL,
  `corruption` DECIMAL(10,3) NULL,
  `negative affect` DECIMAL(10,3) NULL,
  PRIMARY KEY (`hapiness_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table Climat
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS Climat (
  `climat_id` INT NOT NULL AUTO_INCREMENT,
  `date` DATE NULL,
  `average_temperature` DECIMAL(10,3) NULL,
  `average_temperature_uncertainty` DECIMAL(10,3) NULL,
  `country_name` VARCHAR(45) NULL,
  PRIMARY KEY (`climat_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table Terrorism
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS Terrorism (
  `event_id` BIGINT NOT NULL,
  `year` INT NULL,
  `month` INT NULL,
  `day` INT NULL,
  `extended` TINYINT(2) NULL,
  `country_id` INT NULL,
  `country_name` VARCHAR(45) NULL,
  PRIMARY KEY (`event_id`))
ENGINE = InnoDB;
```

DataWarehouse.sql

```
-- -----------------------------------------------------
-- Table dim_climat
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS dim_climat (
  `climat_id` INT NOT NULL AUTO_INCREMENT,
  `average_temperature` DECIMAL(10,3) NULL,
  `average_temperature_uncertainty` DECIMAL(10,3) NULL,
  PRIMARY KEY (`climat_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table dim_terrorism
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS dim_terrorism (
  `event_id` INT NOT NULL AUTO_INCREMENT,
  `event_name` VARCHAR(45) NULL,
  `extended` TINYINT(2) NULL,
  PRIMARY KEY (`event_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table dim_date
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS dim_date (
  `date_id` INT NOT NULL AUTO_INCREMENT,
  `year` INT NULL,
  `month` INT NULL,
  `day` INT NULL,
  PRIMARY KEY (`date_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table dim_country
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS dim_country (
  `country_id` INT NOT NULL AUTO_INCREMENT,
  `country_code` INT NULL,
  `country_name` VARCHAR(45) NULL,
  PRIMARY KEY (`country_id`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table fact_hapiness_analysis
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS fact_hapiness_analysis (
  `fact_hapiness_analysis_id` INT NOT NULL AUTO_INCREMENT,
  `climat_id` INT NULL,
```

```
  `date_id` INT NOT NULL,
  `country_id` INT NOT NULL,
  `event_id` INT NULL,
  `life_ladder` DECIMAL(10,3) NULL,
  `gdp_per_capita` DECIMAL(10,3) NULL,
  `positive_affect` DECIMAL(10,3) NULL,
  `social_support` DECIMAL(10,3) NULL,
  `life_expancy` DECIMAL(10,3) NULL,
  `freedom_choice` DECIMAL(10,3) NULL,
  `generocity` DECIMAL(10,3) NULL,
  `corruption` DECIMAL(10,3) NULL,
  `negative_affect` DECIMAL(10,3) NULL,
  PRIMARY KEY (`fact_hapiness_analysis_id`),
  INDEX `fk1_idx` (`climat_id` ASC) VISIBLE,
  INDEX `fk2_idx` (`date_id` ASC) VISIBLE,
  INDEX `fk3_idx` (`country_id` ASC) VISIBLE,
  INDEX `fk4_idx` (`event_id` ASC) VISIBLE,
  CONSTRAINT `fk1`
    FOREIGN KEY (`climat_id`)
    REFERENCES dim_climat (`climat_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk2`
    FOREIGN KEY (`date_id`)
    REFERENCES dim_date (`date_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk3`
    FOREIGN KEY (`country_id`)
    REFERENCES dim_country (`country_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk4`
    FOREIGN KEY (`event_id`)
    REFERENCES dim_terrorism (`event_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

ETL.sql

```
-- -----------------------------------------------------
-- Table dim_climat
-- -----------------------------------------------------
INSERT INTO DataWarehouse.dim_climat (average_temperature,
average_temperature_uncertainty)
SELECT
  ROUND(average_temperature, 3) as average_temperature,
  ROUND(average_temperature_uncertainty, 3) as
average_temperature_uncertainty
FROM StageZone.Climat;

-- -----------------------------------------------------
-- Table dim_terrorism
-- -----------------------------------------------------
INSERT INTO DataWarehouse.dim_terrorism (event_name, extended)
SELECT event_id, extended
FROM StageZone.Terrorism;

-- -----------------------------------------------------
-- Table dim_country
-- -----------------------------------------------------
INSERT INTO DataWarehouse.dim_country (country_code,
country_name)
SELECT country_id, country_name FROM StageZone.Terrorism
UNION
SELECT NULL, country_name FROM StageZone.Hapiness
UNION
SELECT NULL, country_name FROM StageZone.Climat
WHERE NOT EXISTS (
  SELECT * FROM DataWarehouse.dim_country
  WHERE DataWarehouse.dim_country.country_name =
country_name
);

-- -----------------------------------------------------
-- Table dim_date
-- -----------------------------------------------------
INSERT IGNORE INTO DataWarehouse.dim_date (year, month, day)
SELECT DISTINCT YEAR(date) AS year, MONTH(date) AS month,
DAY(date) AS day
FROM StageZone.Climat
UNION
SELECT DISTINCT year, NULL, NULL FROM StageZone.Hapiness
```

```
UNION
SELECT DISTINCT year, month, day FROM StageZone.Terrorism
WHERE NOT EXISTS (
  SELECT * FROM DataWarehouse.dim_date
  WHERE (DataWarehouse.dim_date.year = year AND
DataWarehouse.dim_date.month = month AND
DataWarehouse.dim_date.day = day)
);
-- -----------------------------------------------------
-- Table fact_hapiness_analysis
-- -----------------------------------------------------
INSERT INTO fact_hapiness_analysis
(climat_id, date_id, country_id, event_id, life_ladder, gdp_per_capita,
positive_affect, social_support, life_expancy, freedom_choice,
generocity, corruption, negative_affect)
select
   CL.climat_id,
  D.date_id,
  CO.country_id,
  T.event_id,
  SH.life_ladder,
  SH.gdp_per_capita,
  SH.positive_affect,
  SH.social_support,
  SH.life_expancy,
  SH.freedom_choice,
  SH.generocity,
  SH.corruption,
  SH.negative_affect
from (
select *
from StageZone.Climat
where YEAR(date) > 2004
) SCL
    inner join dim_climat CL ON CL.average_temperature =
SCL.average_temperature and CL.average_temperature_uncertainty =
SCL.average_temperature_uncertainty
left join StageZone.Terrorism ST on ST.country_name =
SCL.country_name and ST.year = YEAR(SCL.date) and
ST.month = MONTH(SCL.date) and ST.day = DAY(SCL.date)
left join dim_terrorism T on ST.extended = T.extended and
ST.event_name = T.event_name
left join StageZone.Hapiness SH on YEAR(SCL.date) = SH.year and
SH.country_name = SCL.country_name
left join dim_date D on YEAR(SCL.date) = D.year and
```

```
MONTH(SCL.date) = D.month and
DAY(SCL.date) = D.day
left join (select * from dim_country where country_code is null) CO on
CO.country_name = SCL.country_name;
```

CWork.py

```
import pymysql
#Connection with Database
connection = pymysql.connect(
    host="127.0.0.1",
    user="root",
    password="",
    database="DataWarehouse"
)
cursor = connection.cursor()
#Fetching data
cursor = connection.cursor()
query ="""
SELECT f.life_ladder, f.gdp_per_capita, f.social_support,
f.life_expancy, f.freedom_choice,
            c.average_temperature, c.average_temperature_uncertainty,
t.event_name, t.extended
FROM fact_hapiness_analysis AS f
JOIN dim_climat AS c ON f.climat_id = c.climat_id
LEFT JOIN dim_terrorism AS t ON f.event_id = t.event_id
JOIN dim_country AS ct ON f.country_id = ct.country_id
"""
cursor.execute(query)
data = cursor.fetchall()

import pandas as pd
#Preparing data
columns = ['life_ladder', 'gdp_per_capita', 'social_support',
'life_expancy', 'freedom_choice',
           'average_temperature', 'average_temperature_uncertainty',
           'event_name', 'extended']
df = pd.DataFrame(data, columns=columns)

#Adding isTerror instead of event_name
df['isTerror'] = df['event_name'].apply(lambda x: 1 if pd.notnull(x) else
0)
df = df.drop(columns=['event_name'])
```

```
#Analyze structure
df.info()

import numpy as np
#Fixing data
df['extended'] = df['extended'].fillna(0)
noNanData = list(df.columns)[0:5]
for i in noNanData:
    df = df.dropna()
numeric_columns = ['life_ladder', 'gdp_per_capita', 'social_support',
'life_expancy',
                   'freedom_choice', 'average_temperature',
'average_temperature_uncertainty']
df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric,
errors='coerce')

#Analyze structure after fixing
df.info()
df.head(10)

import matplotlib.pyplot as plt
#Visual of relations
fig, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0, 0].plot(df['average_temperature'], df['life_ladder'], 'o')
axs[0, 0].set_xlabel('average_temperature')
axs[0, 0].set_ylabel('life_ladder')
axs[0, 1].plot(df['average_temperature_uncertainty'], df['life_ladder'],
'o')
axs[0, 1].set_xlabel('average_temperature_uncertainty')
axs[0, 1].set_ylabel('life_ladder')
axs[1, 0].plot(df['isTerror'], df['life_ladder'], 'o')
axs[1, 0].set_xlabel('isTerror')
axs[1, 0].set_ylabel('life_ladder')
axs[1, 1].plot(df['extended'], df['life_ladder'], 'o')
axs[1, 1].set_xlabel('extended')
axs[1, 1].set_ylabel('life_ladder')
plt.tight_layout()
plt.show()

#Dividing data
from sklearn.model_selection import train_test_split
X = df[['average_temperature', 'average_temperature_uncertainty',
'isTerror', 'extended']]
y = df['life_ladder']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

```
random_state=0)

import seaborn as sns
#Calculate the correlation matrix
data = pd.concat([y, X], axis=1)
correlation_matrix = data.corr()
#Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error
from sklearn.model_selection import cross_val_score
#Training models
model1 = LinearRegression()
model1.fit(X_train, y_train)
y_pred1 = model1.predict(X_test)
mae1 = mean_absolute_error(y_test, y_pred1)
mse1 = mean_squared_error(y_test, y_pred1)
r2_1 = r2_score(y_test, y_pred1)
scores1 = cross_val_score(model1, X, y, cv=10, scoring='r2')
r2_1_cross_val = scores1.mean()

poly = PolynomialFeatures(degree=4)
X_poly = poly.fit_transform(X)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
model2 = LinearRegression()
model2.fit(X_train_poly, y_train)
y_pred2 = model2.predict(X_test_poly)
mae2 = mean_absolute_error(y_test, y_pred2)
mse2 = mean_squared_error(y_test, y_pred2)
r2_2 = r2_score(y_test, y_pred2)
scores2 = cross_val_score(model2, X_poly, y, cv=10, scoring='r2')
r2_2_cross_val = scores2.mean()

# Compare and select the best model
if r2_1 > r2_2 :
    best_model = model1
    best_model_name = 'Linear Regression'
else:
```

```
    best_model = model2
    best_model_name = 'Polynomial  Regression'

#Finding degree for PR
def find_best_degree(X_train, y_train, X_test, y_test, degrees):
    best_degree = 1
    best_score = -np.inf
    for degree in range(1, degrees+1):
        poly = PolynomialFeatures(degree=degree)
        X_train_poly = poly.fit_transform(X_train)
        X_test_poly = poly.transform(X_test)
        model = LinearRegression()
        model.fit(X_train_poly, y_train)
        score = model.score(X_test_poly, y_test)
        if score > best_score:
            best_score = score
            best_degree = degree

    return best_degree, best_score
best_degree, best_score = find_best_degree(X_train, y_train, X_test,
y_test, degrees=10)
print("Best degree:", best_degree)
print("Best score:", best_score)

# Visualize the Linear Regression predictions
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred1, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Linear Regression Prediction')
plt.show()
# Visualize the Polynomial Regression predictions
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred2, color='green')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Polynomial Regression Prediction Visualization')
plt.show()
# Plotting the predictions and actual both methods values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred1, color='blue', label='Linear Regression')
```

```
plt.scatter(y_test, y_pred2, color='green', label='Polynomial
Regression')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Comparison of Predicted Values')
plt.legend()
plt.show()

#Metrics
print(f"Model 1 - Linear Regression:")
print(f"MSE: {mse1}")
print(f"MAE: {mae1}")
print(f"R^2: {r2_1}")
print(f"10-fold cross-validation: {r2_1_cross_val}")
print()
print(f"Model 2 - Polynomial Linear Regression:")
print(f"MSE: {mse2}")
print(f"MAE: {mae2}")
print(f"R^2: {r2_2}")
print(f"10-fold cross-validation: {r2_2_cross_val}")
print()
# Print the best model
print(f"Best Model: {best_model_name}")

# Print the accuracy for each model
print(f"Model 1 - Linear Regression:")
print(f"Accuracy - testing: {int(model1.score(X_test, y_test)*100)}%")
print()
print(f"Accuracy - training: {int(model1.score(X_train,
y_train)*100)}%")
print()

print(f"Model 2 - Polynomial  Regression:")
print(f"Accuracy - testing: {int(model2.score(X_test_poly,
y_test)*100)}%")
print()
print(f"Accuracy - training: {int(model2.score(X_train_poly,
y_train)*100)}%")
print()

# Close the cursor and connection
cursor.close()
connection.close()
```