```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to
```

Importing the Data

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolu
import shap
import numpy as np
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
ql = pd.read_csv("/content/drive/My Drive/Business Analytic
ql
```

ValueError ValueError •••

Please explain the error:

```
TypeError: unsupported operand typ
```

Hypothesis

The error message "TypeError: unsupported operand type(s) for -: 'NoneType' and 'int'" indicates that you're trying to subtract an integer (10 in this case) from a value that is `None`.

Looking at the traceback and your code, the error occurs in this line:

```
"max_depth": [best_params["max_d
```

Use code with caution

This line is attempting to create a list of values for the `max_depth` parameter in your Grid Search. The problem is that `best_params["max_depth"]` might be `None` (as indicated by the error). This happens when the `RandomizedSearchCV` in the previous step determined that the best value for `max_depth` is `None` (which means unlimited depth for the Random Forest).

Suggested Changes

```
param_grid = {
    "n_estimators": [best_params
    "max_depth": [None, best_par
    "min_samples_split": [best_p
    "min_samples_leaf": [best_pa
    "max_features": [best_params
```

	country	Purchasing Power Value	Purchasing Power Category	Safety Value	Safe Catego
0	Afghanistan	32.15	'Very Low'	25.33	'L
1	Aland Islands	125.01	'Very High'	71.81	'Hi
2	Albania	42.82	'Low'	55.52	'Modera
3	Alderney	0.00	NaN	83.79	'V Hi
4	Algeria	27.60	'Very Low'	47.54	'Modera
...	...	...	...	...	
231	Wallis And Futuna	0.00	NaN	0.00	N
232	Western Sahara	0.00	NaN	62.87	'Hi
233	Yemen	20.74	'Very Low'	34.07	'L
234	Zambia	22.32	'Very Low'	54.39	'Modera
235	Zimbabwe	28.76	'Very Low'	39.31	'L

236 rows × 19 columns

Next steps: Generate code with ql | View recommended plots | N

Data Preprocessing

```
ql.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 236 entries, 0 to 235
Data columns (total 19 columns):
 #   Column                              Non-Null Count
---  ------                              --------------
 0   country                             236 non-null
 1   Purchasing Power Value              236 non-null
 2   Purchasing Power Category           190 non-null
 3   Safety Value                        236 non-null
 4   Safety Category                     234 non-null
```

```
    "bootstrap": [best_params["b
}
```

Use code with caution

Explanation of Changes:

1. **Conditional Logic for `max_depth`:**
   - We use an inline `if` statement to handle the case where `best_params["max_depth"` is `None`.
   - If `best_params["max_depth"` is `None`, the list will include `None`, `best_params["max_depth"]` `None`.
   - If it's not `None`, it'll perform the addition (`+ 10`), ensuring the code doesn't try to subtract from `None`.

▶ Sources

Rate this answer

```
 5   Health Care Value                  236 non-null
 6   Health Care Category               221 non-null
 7   Climate Value                      236 non-null
 8   Climate Category                   114 non-null
 9   Cost of Living Value               236 non-null
 10  Cost of Living Category            191 non-null
 11  Property Price to Income Value     236 non-null
 12  Property Price to Income Category  215 non-null
 13  Traffic Commute Time Value         236 non-null
 14  Traffic Commute Time Category      207 non-null
 15  Pollution Value                    236 non-null
 16  Pollution Category                 226 non-null
 17  Quality of Life Value              236 non-null
 18  Quality of Life Category           114 non-null
dtypes: float64(7), object(12)
memory usage: 35.2+ KB
```

```
ql['Quality of Life Value'] = ql['Quality of Life Value'].s
# Removing '' from data in categorical column
categorical_columns = ['Purchasing Power Category', 'Safety
                       'Climate Category', 'Cost of Living
                       'Traffic Commute Time Category', 'Pc
for col in categorical_columns:
    ql[col] = ql[col].str.replace("'", "", regex=True)
ql
```

	country	Purchasing Power Value	Purchasing Power Category	Safety Value	Safe Catego
0	Afghanistan	32.15	Very Low	25.33	L
1	Aland Islands	125.01	Very High	71.81	Hi
2	Albania	42.82	Low	55.52	Modera
3	Alderney	0.00	NaN	83.79	Very Hi
4	Algeria	27.60	Very Low	47.54	Modera
...	...	...	...	...	
231	Wallis And Futuna	0.00	NaN	0.00	N
232	Western Sahara	0.00	NaN	62.87	Hi
233	Yemen	20.74	Very Low	34.07	L
234	Zambia	22.32	Very Low	54.39	Modera
235	Zimbabwe	28.76	Very Low	39.31	L

236 rows × 19 columns

Next steps: Generate code with ql View recommended plots N

```
ql['Property Price to Income Value'] = pd.to_numeric(ql['Pr
ql['Quality of Life Value'] = pd.to_numeric(ql['Quality of
```

```
ql.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 236 entries, 0 to 235
Data columns (total 19 columns):
 #   Column                          Non-Null Count
---  ------                          --------------
 0   country                         236 non-null
 1   Purchasing Power Value          236 non-null
 2   Purchasing Power Category       190 non-null
 3   Safety Value                    236 non-null
 4   Safety Category                 234 non-null
 5   Health Care Value               236 non-null
 6   Health Care Category            221 non-null
 7   Climate Value                   236 non-null
 8   Climate Category                114 non-null
 9   Cost of Living Value            236 non-null
 10  Cost of Living Category         191 non-null
 11  Property Price to Income Value  233 non-null
```

```
 12  Property Price to Income Category  215 non-null
 13  Traffic Commute Time Value         236 non-null
 14  Traffic Commute Time Category      207 non-null
 15  Pollution Value                    236 non-null
 16  Pollution Category                 226 non-null
 17  Quality of Life Value              236 non-null
 18  Quality of Life Category           114 non-null
dtypes: float64(9), object(10)
memory usage: 35.2+ KB
```

```
ql.head()
```

	country	Purchasing Power Value	Purchasing Power Category	Safety Value	Safety Category
0	Afghanistan	32.15	Very Low	25.33	Low
1	Aland Islands	125.01	Very High	71.81	High
2	Albania	42.82	Low	55.52	Moderate
3	Alderney	0.00	NaN	83.79	Very High
4	Algeria	27.60	Very Low	47.54	Moderate

Next steps: Generate code with ql View recommended plots N

Data Cleaning

```
ql.isnull().sum()
```

	0
country	0
Purchasing Power Value	0
Purchasing Power Category	46
Safety Value	0
Safety Category	2
Health Care Value	0
Health Care Category	15
Climate Value	0
Climate Category	122
Cost of Living Value	0
Cost of Living Category	45
Property Price to Income Value	3
Property Price to Income Category	21
Traffic Commute Time Value	0
Traffic Commute Time Category	29
Pollution Value	0
Pollution Category	10
Quality of Life Value	0
Quality of Life Category	122

Using KNN Imputer to handling missing value in numerical and categorical variable

```
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Select the numerical and categorical columns to be impute
numeric_columns = ['Purchasing Power Value', 'Safety Value'
                   'Climate Value', 'Cost of Living Value',
                   'Traffic Commute Time Value', 'Pollution

# Scaling the numerical data
scaler = StandardScaler()
ql[numeric_columns] = scaler.fit_transform(ql[numeric_colum
```

```
# Using KNN Imputer to replace missing values in numeric cc
imputer_numeric = KNNImputer(n_neighbors=5)  # Menggunakan
ql[numeric_columns] = imputer_numeric.fit_transform(ql[nume

# Impute missing values in a categorical column with mode (
imputer_categorical = SimpleImputer(strategy='most_frequent
ql[categorical_columns] = imputer_categorical.fit_transform

# After imputation, we return the numerical data to their c
ql[numeric_columns] = scaler.inverse_transform(ql[numeric_c

# Checking the imputation results
ql
```

	country	Purchasing Power Value	Purchasing Power Category	Safety Value	Safe Catego
0	Afghanistan	32.15	Very Low	25.33	L
1	Aland Islands	125.01	Very High	71.81	Hi
2	Albania	42.82	Low	55.52	Modera
3	Alderney	0.00	Very Low	83.79	Very Hi
4	Algeria	27.60	Very Low	47.54	Modera
...	...	...	...	...	
231	Wallis And Futuna	0.00	Very Low	0.00	Modera
232	Western Sahara	0.00	Very Low	62.87	Hi
233	Yemen	20.74	Very Low	34.07	L
234	Zambia	22.32	Very Low	54.39	Modera
235	Zimbabwe	28.76	Very Low	39.31	L

236 rows × 19 columns

Next steps: Generate code with ql View recommended plots N

Feature Engineering

```
category_mapping = {'Very Low': 1, 'Low': 2, 'Moderate': 3,
columns_to_map = ['Purchasing Power Category', 'Safety Cate
```

```
                  'Climate Category', 'Cost of Living Categ
                  'Traffic Commute Time Category', 'Polluti

for col in columns_to_map:
    ql[col] = ql[col].map(category_mapping)
ql
```

	country	Purchasing Power Value	Purchasing Power Category	Safety Value	Safe Catego
0	Afghanistan	32.15	1	25.33	
1	Aland Islands	125.01	5	71.81	
2	Albania	42.82	2	55.52	
3	Alderney	0.00	1	83.79	
4	Algeria	27.60	1	47.54	
...	...	...	...	...	
231	Wallis And Futuna	0.00	1	0.00	
232	Western Sahara	0.00	1	62.87	
233	Yemen	20.74	1	34.07	
234	Zambia	22.32	1	54.39	
235	Zimbabwe	28.76	1	39.31	

236 rows × 19 columns

Next steps: Generate code with ql View recommended plots N

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

ql['country'] = label_encoder.fit_transform(ql['country'])
ql
```

	country	Purchasing Power Value	Purchasing Power Category	Safety Value	Safety Category
0	0	32.15	1	25.33	2
1	1	125.01	5	71.81	4
2	2	42.82	2	55.52	3
3	3	0.00	1	83.79	5
4	4	27.60	1	47.54	3
...	...	...	...	...	...
231	231	0.00	1	0.00	3
232	232	0.00	1	62.87	4
233	233	20.74	1	34.07	2
234	234	22.32	1	54.39	3
235	235	28.76	1	39.31	2

236 rows × 19 columns

Next steps: Generate code with ql View recommended plots N

```
# Adding additional features

ql['Income_to_Property_Ratio'] = ql['Purchasing Power Value
ql['Safety_to_Pollution_Ratio'] = ql['Safety Value'] / (ql[
ql['Health_Index_Score'] = (ql['Health Care Value'] + ql['C

ql.head()
```

	country	Purchasing Power Value	Purchasing Power Category	Safety Value	Safety Category	H
0	0	32.15	1	25.33	2	
1	1	125.01	5	71.81	4	
2	2	42.82	2	55.52	3	
3	3	0.00	1	83.79	5	
4	4	27.60	1	47.54	3	

5 rows × 22 columns

Scaling Numerical Data

```
scaler = StandardScaler()
ql[numeric_columns] = scaler.fit_transform(ql[numeric_colum
ql
```

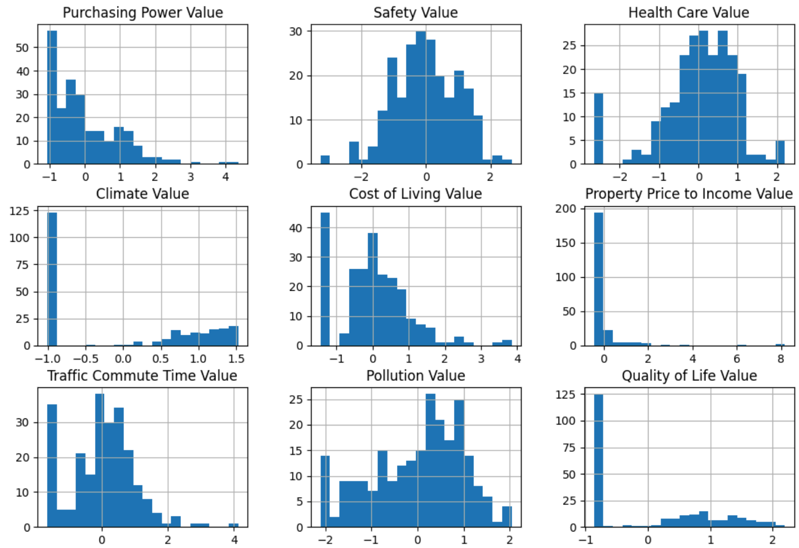
	country	Purchasing Power Value	Purchasing Power Category	Safety Value	Safe Catego
0	0	-0.451334	1	-1.774126	
1	1	1.337947	5	0.979686	
2	2	-0.245738	2	0.014548	
3	3	-1.070819	1	1.689468	
4	4	-0.539006	1	-0.458245	
...	...	...	...	...	
231	231	-1.070819	1	-3.274858	
232	232	-1.070819	1	0.450015	
233	233	-0.671189	1	-1.256305	
234	234	-0.640744	1	-0.052401	
235	235	-0.516655	1	-0.945849	

236 rows × 22 columns

EDA (Exploratory Data Analysis)

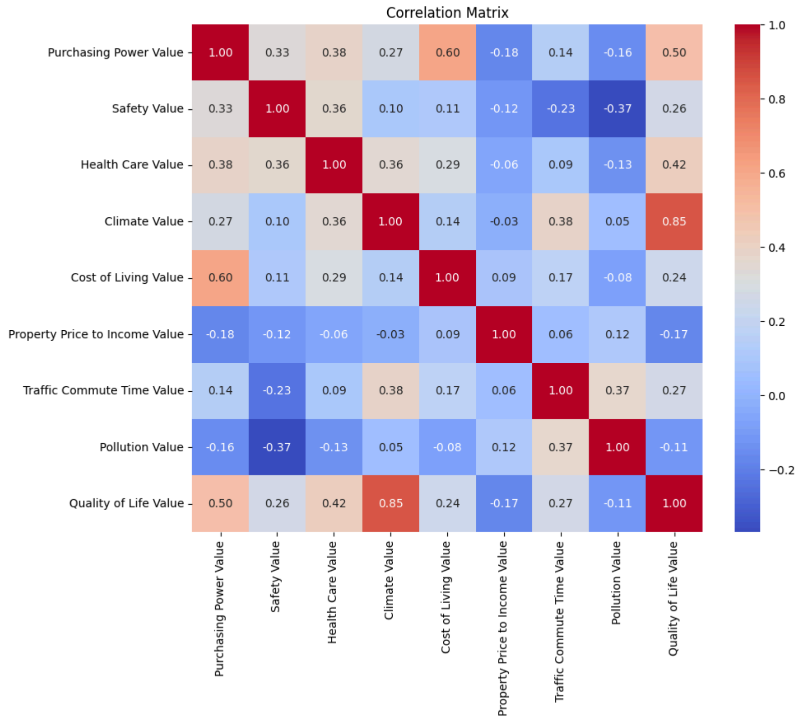
Data Distribution

```
ql[numeric_columns].hist(figsize=(12, 8), bins=20)
plt.show()
```



Correlation

```
plt.figure(figsize=(10, 8))
sns.heatmap(ql[numeric_columns].corr(), annot=True, cmap="c
plt.title("Correlation Matrix")
plt.show()
```

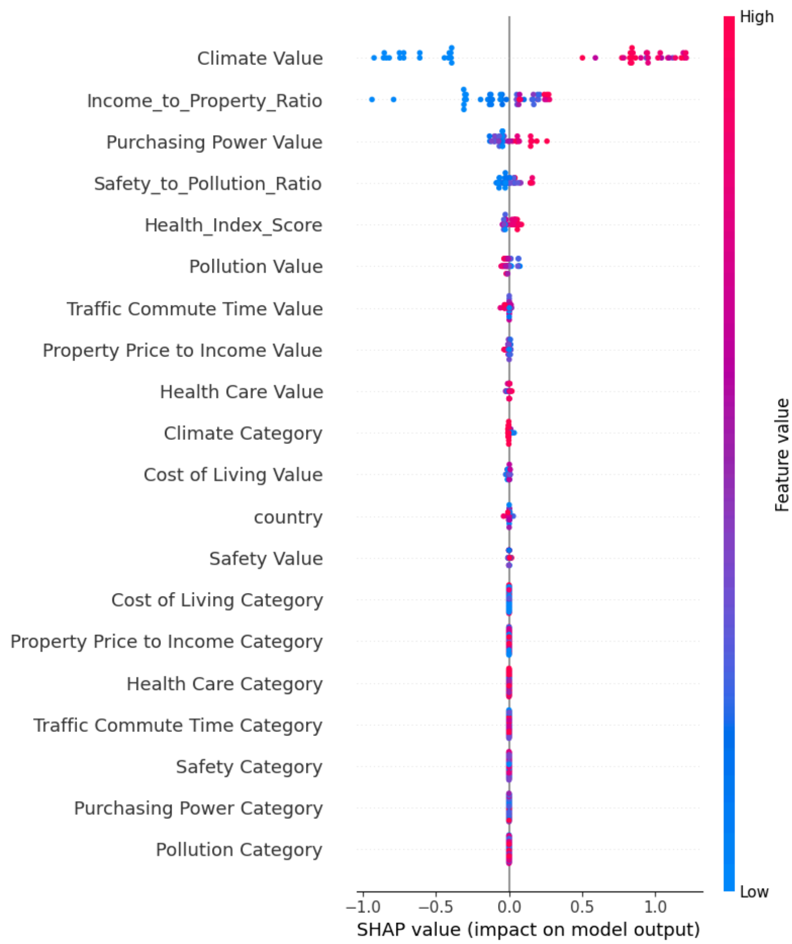


Model Training and Machine Learning

```
# Using Quality of Life as a target for regression
X = ql.drop(columns=['Quality of Life Value', 'Quality of Li
y = ql['Quality of Life Value']

# PCA for Dimensionality Reduction
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X)

# Split the Data
```

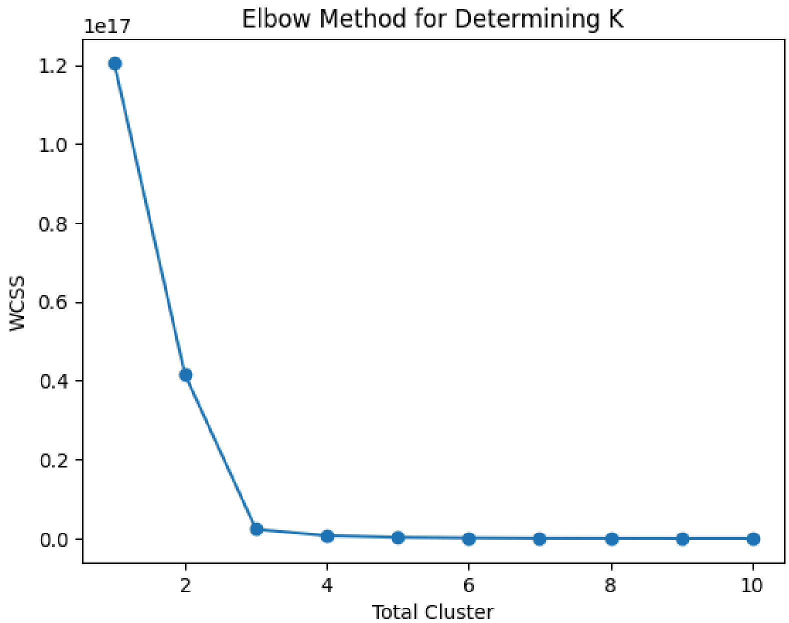
```
X_train, X_test, y_train, y_test = train_test_split(X, y, te

# Model Evaluation Function
def evaluate_model(model, X_train, X_test, y_train, y_test, 
    y_pred = model.predict(X_test)
    rmse = mean_squared_error(y_test, y_pred) ** 0.5
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    cv_r2 = cross_val_score(model, X_train, y_train, cv=5, s

    print(f"{model_name} Performance:")
    print(f"RMSE: {rmse:.4f}, MAE: {mae:.4f}, R²: {r2:.4f}, 

    return y_pred

# 1 Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = evaluate_model(lr, X_train, X_test, y_train, y_t

# 2 Random Forest Regressor with GridSearchCV
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [None, 20, 30],
    'min_samples_split': [2, 5]
}
rf = GridSearchCV(RandomForestRegressor(random_state=42), pa
rf.fit(X_train, y_train)
y_pred_rf = evaluate_model(rf.best_estimator_, X_train, X_te

# 3 XGBoost Regressor with SHAP Feature Importance
xgb = XGBRegressor(objective="reg:squarederror", random_stat
xgb.fit(X_train, y_train)
y_pred_xgb = evaluate_model(xgb, X_train, X_test, y_train, y

# SHAP Analysis
explainer = shap.Explainer(xgb)
shap_values = explainer(X_test)
shap.summary_plot(shap_values, X_test)
```

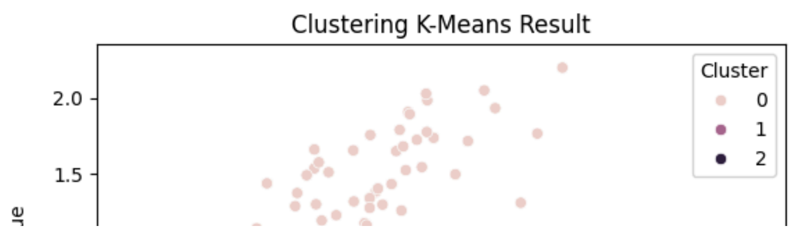
```
Linear Regression Performance:
RMSE: 0.3180, MAE: 0.2272, R²: 0.8988, CV R²: -11072451

Random Forest Performance:
RMSE: 0.1479, MAE: 0.0927, R²: 0.9781, CV R²: 0.9483

XGBoost Performance:
RMSE: 0.1523, MAE: 0.0855, R²: 0.9768, CV R²: 0.9447
```



Clustering using K-Means with PCA-Reduced Data

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42, n_init=1
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss, marker='o')
plt.xlabel('Total Cluster')
plt.ylabel('WCSS')
plt.title('Elbow Method for Determining K')
plt.show()

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
ql['Cluster'] = kmeans.fit_predict(X_pca)

sns.scatterplot(x=ql['Purchasing Power Value'], y=ql['Quali
plt.title("Clustering K-Means Result")
plt.show()
```





Hyperparameter Tuning

```
from sklearn.model_selection import RandomizedSearchCV, Grid
import shap

# Hyperparameter Grid for Randomized Search
param_dist = {
    "n_estimators": [50, 100, 200, 300],
    "max_depth": [None, 10, 20, 30, 50],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 5, 10],
    "max_features": ["sqrt", "log2", None],
    "bootstrap": [True, False]
}
```

Enter a prompt here

0 / 2000