Lab 8: Define and Solve an ML Problem of Your Choosing

In [2]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In this lab assignment, you will follow the machine learning life cycle and implement a model to solve a machine learning problem of your choosing. You will select a data set and choose a predictive problem that the data set supports. You will then inspect the data with your problem in mind and begin to formulate a project plan. You will then implement the machine learning project plan.

You will complete the following tasks:

1. Build Your DataFrame
2. Define Your ML Problem
3. Perform exploratory data analysis to understand your data.
4. Define Your Project Plan
5. Implement Your Project Plan:
    - Prepare your data for your model.
    - Fit your model to the training data and evaluate your model.
    - Improve your model's performance.

Part 1: Build Your DataFrame

You will have the option to choose one of four data sets that you have worked with in this program:

- The "census" data set that contains Census information from 1994: `censusData.csv`
- Airbnb NYC "listings" data set: `airbnbListingsData.csv`
- World Happiness Report (WHR) data set: `WHR2018Chapter2OnlineData.csv`
- Book Review data set: `bookReviewsData.csv`

Note that these are variations of the data sets that you have worked with in this program. For example, some do not include some of the preprocessing necessary for specific models.

Load a Data Set and Save it as a Pandas DataFrame

The code cell below contains filenames (path + filename) for each of the four data sets available to you.

Task: In the code cell below, use the same method you have been using to load the data using `pd.read_csv()` and save it to DataFrame `df`.

You can load each file as a new DataFrame to inspect the data before choosing your data set.

In [3]:
```
# File names of the four data sets
adultDataSet_filename = os.path.join(os.getcwd(), "data", "censusData.csv")
airbnbDataSet_filename = os.path.join(os.getcwd(), "data", "airbnbListingsData.csv")
WHRDataSet_filename = os.path.join(os.getcwd(), "data", "WHR2018Chapter2OnlineData.csv")
bookReviewDataSet_filename = os.path.join(os.getcwd(), "data", "bookReviewsData.csv")


df = pd.read_csv(WHRDataSet_filename)

df.head()
```

Out[3]:

	country	year	Life Ladder	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption
0	Afghanistan	2008	3.723590	7.168690	0.450662	49.209663	0.718114	0.181819	0.881686
1	Afghanistan	2009	4.401778	7.333790	0.552308	49.624432	0.678896	0.203614	0.850035
2	Afghanistan	2010	4.758381	7.386629	0.539075	50.008961	0.600127	0.137630	0.706766
3	Afghanistan	2011	3.831719	7.415019	0.521104	50.367298	0.495901	0.175329	0.731109
4	Afghanistan	2012	3.782938	7.517126	0.520637	50.709263	0.530935	0.247159	0.775620

Part 2: Define Your ML Problem

Next you will formulate your ML Problem. In the markdown cell below, answer the following questions:

1. List the data set you have chosen.
2. What will you be predicting? What is the label?
3. Is this a supervised or unsupervised learning problem? Is this a clustering, classification or regression problem? Is it a binary classificaiton or multi-class classifiction problem?
4. What are your features? (note: this list may change after your explore your data)
5. Explain why this is an important problem. In other words, how would a company create value with a model that predicts this label?

My ML problem

1. WHR2018Chapter2OnlineData.csv (World Happiness Report 2018 dataset)
2. Life Ladder score from 1-10 (respondents' measure of their own happiness)
3. Supervised learning; Regression problem
4. Other columns in the df, such as Log GDP per capita, Social support, Healthy life expectancy at birth. Freedom to make life choices, Generosity, Perceptions of corruption, etc.
5. This problem is important because it helps us understand how to make people happier, which is crucial for government policy making, company management, community building, etc.

Part 3: Understand Your Data

The next step is to perform exploratory data analysis. Inspect and analyze your data set with your machine learning problem in mind. Consider the following as you inspect your data:

1. What data preparation techniques would you like to use? These data preparation techniques may include:
    - addressing missingness, such as replacing missing values with means
    - finding and replacing outliers
    - renaming features and labels
    - finding and replacing outliers
    - performing feature engineering techniques such as one-hot encoding on categorical features
    - selecting appropriate features and removing irrelevant features
    - performing specific data cleaning and preprocessing techniques for an NLP problem
    - addressing class imbalance in your data sample to promote fair AI
2. What machine learning model (or models) you would like to use that is suitable for your predictive problem and data?
    - Are there other data preparation techniques that you will need to apply to build a balanced modeling data set for your problem and model? For example, will you need to scale your data?
3. How will you evaluate and improve the model's performance?
    - Are there specific evaluation metrics and methods that are appropriate for your model?

Think of the different techniques you have used to inspect and analyze your data in this course. These include using Pandas to apply data filters, using the Pandas `describe()` method to get insight into key statistics for each column, using the Pandas `dtypes` property to inspect the data type of each column, and using Matplotlib and Seaborn to detect outliers and visualize relationships between features and labels. If you are working on a classification problem, use techniques you have learned to determine if there is class imbalance.

Task: Use the techniques you have learned in this course to inspect and analyze your data. You can import additional packages that you have used in this course that you will need to perform this task.

Note: You can add code cells if needed by going to the **Insert** menu and clicking on **Insert Cell Below** in the drop-drown menu.

In [4]:
```
print(df.shape)
df.isnull().sum()
```

```
(1562, 19)
```

Out[4]:
```
country                                                       0
year                                                          0
Life Ladder                                                   0
Log GDP per capita                                           27
Social support                                               13
Healthy life expectancy at birth                              9
Freedom to make life choices                                 29
Generosity                                                   80
Perceptions of corruption                                    90
Positive affect                                              18
Negative affect                                              12
Confidence in national government                           161
Democratic Quality                                          171
Delivery Quality                                            171
Standard deviation of ladder by country-year                  0
Standard deviation/Mean of ladder by country-year             0
GINI index (World Bank estimate)                            979
GINI index (World Bank estimate), average 2000-15           176
gini of household income reported in Gallup, by wp5-year    357
dtype: int64
```

In [5]:
```
df.describe()
```

Out[5]:

	year	Life Ladder	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosi
count	1562.000000	1562.000000	1535.000000	1549.000000	1553.000000	1533.000000	1482.0000
mean	2011.820743	5.433676	9.220822	0.810669	62.249887	0.728975	0.0000
std	3.419787	1.121017	1.184035	0.119370	7.960671	0.145408	0.1642
min	2005.000000	2.661718	6.377396	0.290184	37.766476	0.257534	-0.3229
25%	2009.000000	4.606351	8.310665	0.748304	57.299580	0.633754	-0.1143
50%	2012.000000	5.332600	9.398610	0.833047	63.803192	0.748014	-0.0226
75%	2015.000000	6.271025	10.190634	0.904329	68.098228	0.843628	0.0946
max	2017.000000	8.018934	11.770276	0.987343	76.536362	0.985178	0.6777

Part 4: Define Your Project Plan

Now that you understand your data, in the markdown cell below, define your plan to implement the remaining phases of the machine learning life cycle (data preparation, modeling, evaluation) to solve your ML problem. Answer the following questions:

- Do you have a new feature list? If so, what are the features that you chose to keep and remove after inspecting the data?
- Explain different data preparation techniques that you will use to prepare your data for modeling.
- What is your model (or models)?

- Describe your plan to train your model, analyze its performance and then improve the model. That is, describe your model building, validation and selection plan to produce a model that generalizes well to new data.

1. Selected features:
    - Log GDP per capita
    - Social support
    - Healthy life expectancy at birth
    - Freedom to make life choices
    - Generosity
    - Perceptions of corruption
    - Positive affect
    - Negative affect
2. Data preparation techniques I will use:
    - feature engineering: select the most relevant features using correlation matrix
    - handling missing values: replace by country average, drop if there is still missing value
    - scaling: standardize numerical features to ensure they are on a similar scale.
3. Machine Learning Models to use:
    - Linear Regression (LR)
    - Decision Tree Regressor (DT)
    - Random Forest Regressor (RF)
    - Gradient Boosting Regressor (GBDT)
    - Stacking Regressor (Stacking)
4. Model Training, Evaluation, and Improvement:
    - Training: Train-test split (80-20)
    - Evaluation Metrics: MAE, MSE, R^2.
    - Improvement: Grid Search, Ensemble Learning.

Part 5: Implement Your Project Plan

Task: In the code cell below, import additional packages that you have used in this course that you will need to implement your project plan.

In [6]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, StackingR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Task: Use the rest of this notebook to carry out your project plan.

You will:

1. Prepare your data for your model.
2. Fit your model to the training data and evaluate your model.

3. Improve your model's performance by performing model selection and/or feature selection techniques to find best model for your problem.

Add code cells below and populate the notebook with commentary, code, analyses, results, and figures as you see fit.

Important Additional Knowledge: meaning of each columns in the World Happiness Report

- **Life Ladder:** Overall life satisfaction on a scale from 0 to 10.
- **Log GDP per capita:** Natural logarithm of GDP per person, indicating economic output.
- **Social support:** Availability of help from family, friends, or community.
- **Healthy life expectancy at birth:** Expected years of healthy life for a newborn.
- **Freedom to make life choices:** Personal autonomy in making important life decisions.
- **Generosity:** Charitable behavior, including donations and volunteer work.
- **Perceptions of corruption:** Perceived level of corruption in government and business.
- **Positive affect:** Frequency of experiencing positive emotions like happiness.
- **Negative affect:** Frequency of experiencing negative emotions like sadness.
- **Confidence in national government:** Trust in the national government.
- **Democratic Quality:** Quality of democratic processes and political participation.
- **Delivery Quality:** Effectiveness of public services and government institutions.
- **Standard deviation of ladder by country-year:** Variability of Life Ladder scores within a country-year.
- **Standard deviation/Mean of ladder by country-year:** Normalized measure of disparity in life satisfaction.
- **GINI index (World Bank estimate):** Income inequality measure, 0 (equality) to 1 (max inequality).
- **GINI index (World Bank estimate), average 2000-15:** Average GINI index from 2000 to 2015.
- **Gini of household income reported in Gallup:** Income inequality from Gallup household data.

Data Preparation

In [7]:
```
#df = df.groupby('country').mean().reset_index()
#df = df.drop(columns=['year'])
#df
```

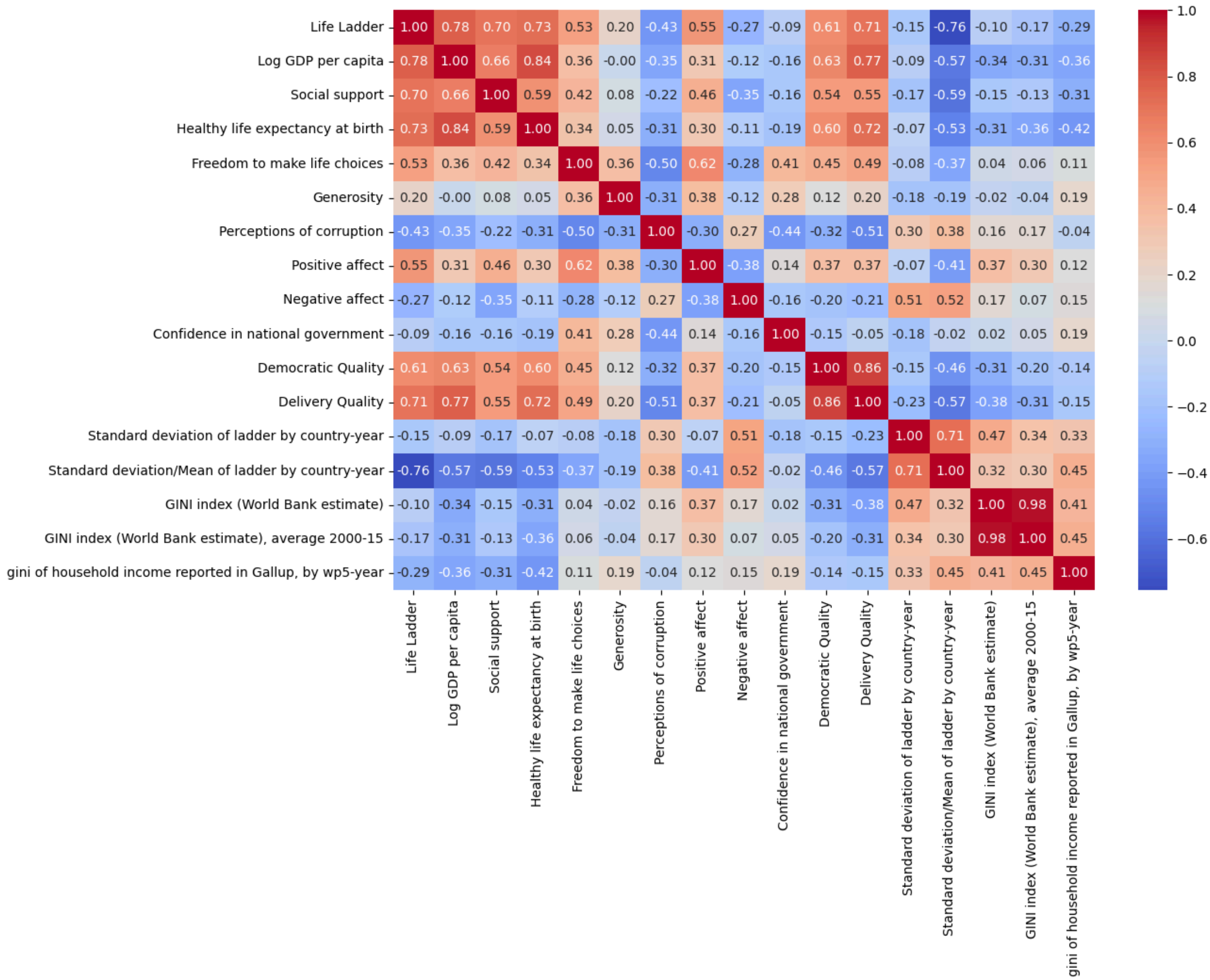
In [8]:
```
#print(df.shape)
#df.isnull().sum()
```

In [9]:
```
#df.to_csv("WHR2018byCountry.csv", index=False)
```

Feature Engineering: selection using correlation matrix

Firstly we use a correlation matrix to determine which columns to drop.

In [10]:
```
df = df.drop(columns=['year'])
corr_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
```



Columns to drop:

- Standard deviation/Mean of ladder by country-year: SD/Mean is a better metric that serves same purpose
- GINI index (World Bank estimate): too many missing values, and gallup gini serves same purpose
- GINI index (World Bank estimate), average 2000-15: same reason as above
- Confidence in national government: many missing values, also lowest correlation with life ladder (happiness)
- Democratic Quality: very high correlation with delivery quality
- Healthy life expectancy at birth: very high correlation with log GDP, and likely a result of it

In [11]:
```
columns_to_drop = [
    'Standard deviation of ladder by country-year',
    'GINI index (World Bank estimate)',
    'GINI index (World Bank estimate), average 2000-15',
    'Confidence in national government',
    'Democratic Quality',
    'Healthy life expectancy at birth'
]
df = df.drop(columns=columns_to_drop)
df.columns
```

```
Out[11]: Index(['country', 'Life Ladder', 'Log GDP per capita', 'Social support',
       'Freedom to make life choices', 'Generosity',
       'Perceptions of corruption', 'Positive affect', 'Negative affect',
       'Delivery Quality', 'Standard deviation/Mean of ladder by country-year',
       'gini of household income reported in Gallup, by wp5-year'],
      dtype='object')
```

```
In [12]: df.rename(columns={
    'Life Ladder': 'happiness_score',
    'Log GDP per capita': 'log_GDP',
    'Social support': 'social_support',
    'Freedom to make life choices': 'freedom_of_choice',
    'Generosity': 'generosity',
    'Perceptions of corruption': 'corruption',
    'Positive affect': 'positive_emotions',
    'Negative affect': 'negative_emotions',
    'Delivery Quality': 'policy_delivery',
    'Standard deviation/Mean of ladder by country-year': 'variation_in_happiness',
    'gini of household income reported in Gallup, by wp5-year': 'gallup_gini'
}, inplace=True)
df.columns
```

```
Out[12]: Index(['country', 'happiness_score', 'log_GDP', 'social_support',
       'freedom_of_choice', 'generosity', 'corruption', 'positive_emotions',
       'negative_emotions', 'policy_delivery', 'variation_in_happiness',
       'gallup_gini'],
      dtype='object')
```

Handling Missing Data: replace with country average

```
In [13]: print(df.shape)
df.isnull().sum()
```

```
(1562, 12)
```

```
Out[13]: country                    0
happiness_score            0
log_GDP                   27
social_support            13
freedom_of_choice         29
generosity                80
corruption                90
positive_emotions         18
negative_emotions         12
policy_delivery          171
variation_in_happiness     0
gallup_gini              357
dtype: int64
```

We notice that there are still many missing values that we need to handle. My approach is:

- identify the missing value in each row
- if the country average data across years is available, replace the missing value with country's average
- if not, drop the row

```
In [14]: country_means = df.groupby('country').transform('mean')
df = df.fillna(country_means)
```

```
df.isnull().sum()
```

Out[14]:
```
country                   0
happiness_score           0
log_GDP                  12
social_support            1
freedom_of_choice         0
generosity               13
corruption               22
positive_emotions         1
negative_emotions         0
policy_delivery          32
variation_in_happiness    0
gallup_gini               6
dtype: int64
```

In [15]:
```
df.dropna(inplace=True)

print(df.shape)
df.isnull().sum()
```

```
(1500, 12)
```

Out[15]:
```
country                   0
happiness_score           0
log_GDP                   0
social_support            0
freedom_of_choice         0
generosity                0
corruption                0
positive_emotions         0
negative_emotions         0
policy_delivery           0
variation_in_happiness    0
gallup_gini               0
dtype: int64
```

Standardization: using scaler

the last step before model training is to use scaler to standardize our numerical data.

In [16]:
```
features = df.drop(columns=['happiness_score', 'country'])
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

features_scaled = pd.DataFrame(features_scaled, columns=features.columns)
features_scaled.describe()
```

Out[16]:

	log_GDP	social_support	freedom_of_choice	generosity	corruption	positive_e
count	1.500000e+03	1.500000e+03	1.500000e+03	1.500000e+03	1.500000e+03	1.5000
mean	-3.031649e-16	-9.379164e-16	-3.102703e-16	9.473903e-18	-4.026409e-17	1.795
std	1.000334e+00	1.000334e+00	1.000334e+00	1.000334e+00	1.000334e+00	1.0003
min	-2.382467e+00	-4.333920e+00	-3.277081e+00	-1.983907e+00	-3.741942e+00	-3.219
25%	-7.810398e-01	-5.257369e-01	-6.623738e-01	-6.885285e-01	-3.072137e-01	-8.107
50%	1.563332e-01	2.050996e-01	1.194860e-01	-1.420441e-01	2.992784e-01	8.468
75%	8.310491e-01	7.774228e-01	8.000938e-01	5.725364e-01	6.861733e-01	8.425
max	2.119613e+00	1.463280e+00	1.756386e+00	4.134306e+00	1.246631e+00	2.178

Model Building

In [17]:
```
X = features_scaled
y = df['happiness_score']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
X_train.shape, X_test.shape
```

Out[17]: ((1200, 10), (300, 10))

We will train 5 different models:

- Linear Regression (LR)
- Decision Tree Regressor (DT)
- Stacking Regressor (ST, by stacking LR and DT)
- Random Forest Regressor (RF)
- Gradient Boosting Regressor (GB)

In [18]:
```
lr = LinearRegression()
dt = DecisionTreeRegressor(random_state=42)
rf = RandomForestRegressor(random_state=42)
gb = GradientBoostingRegressor(random_state=42)
st = StackingRegressor(estimators=[('lr', lr),('dt', dt),])

models = {
    'Linear Regression': lr,
    'Decision Tree': dt,
    'Stacking Regressor': st,
    'Random Forest': rf,
    'Gradient Boosting': gb,
}

model_names = []
mse_scores = []
r2_scores = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
```

```
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'{name} - MAE: {mae:.4f}, MSE: {mse:.4f}, R2: {r2:.4f}')
    mse_scores.append(mse)
    r2_scores.append(r2)
    model_names.append(name)
```

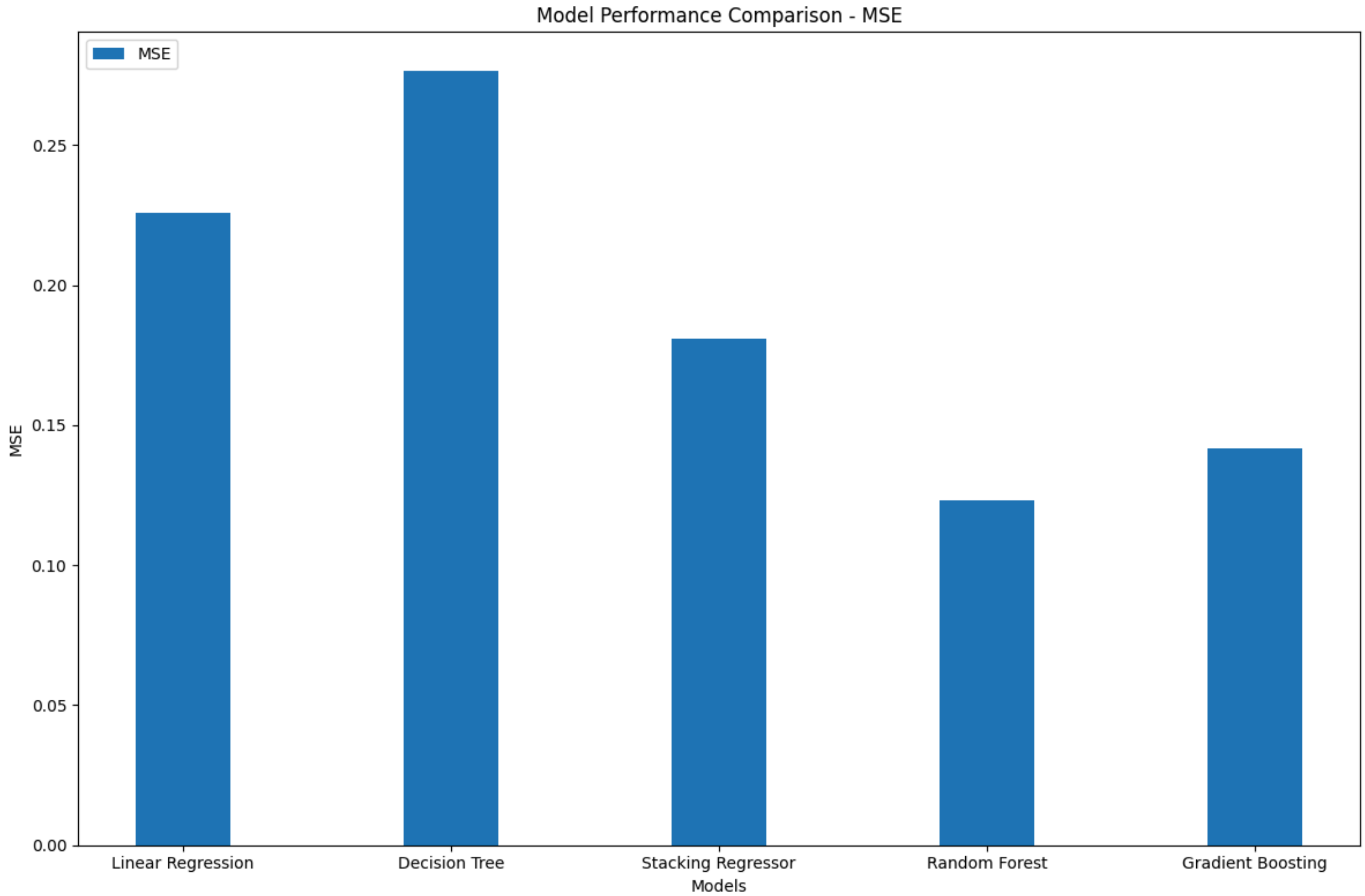
```
Linear Regression - MAE: 0.3714, MSE: 0.2256, R2: 0.8368
Decision Tree - MAE: 0.3803, MSE: 0.2766, R2: 0.8000
Stacking Regressor - MAE: 0.3225, MSE: 0.1808, R2: 0.8692
Random Forest - MAE: 0.2595, MSE: 0.1231, R2: 0.9110
Gradient Boosting - MAE: 0.2806, MSE: 0.1415, R2: 0.8976
```

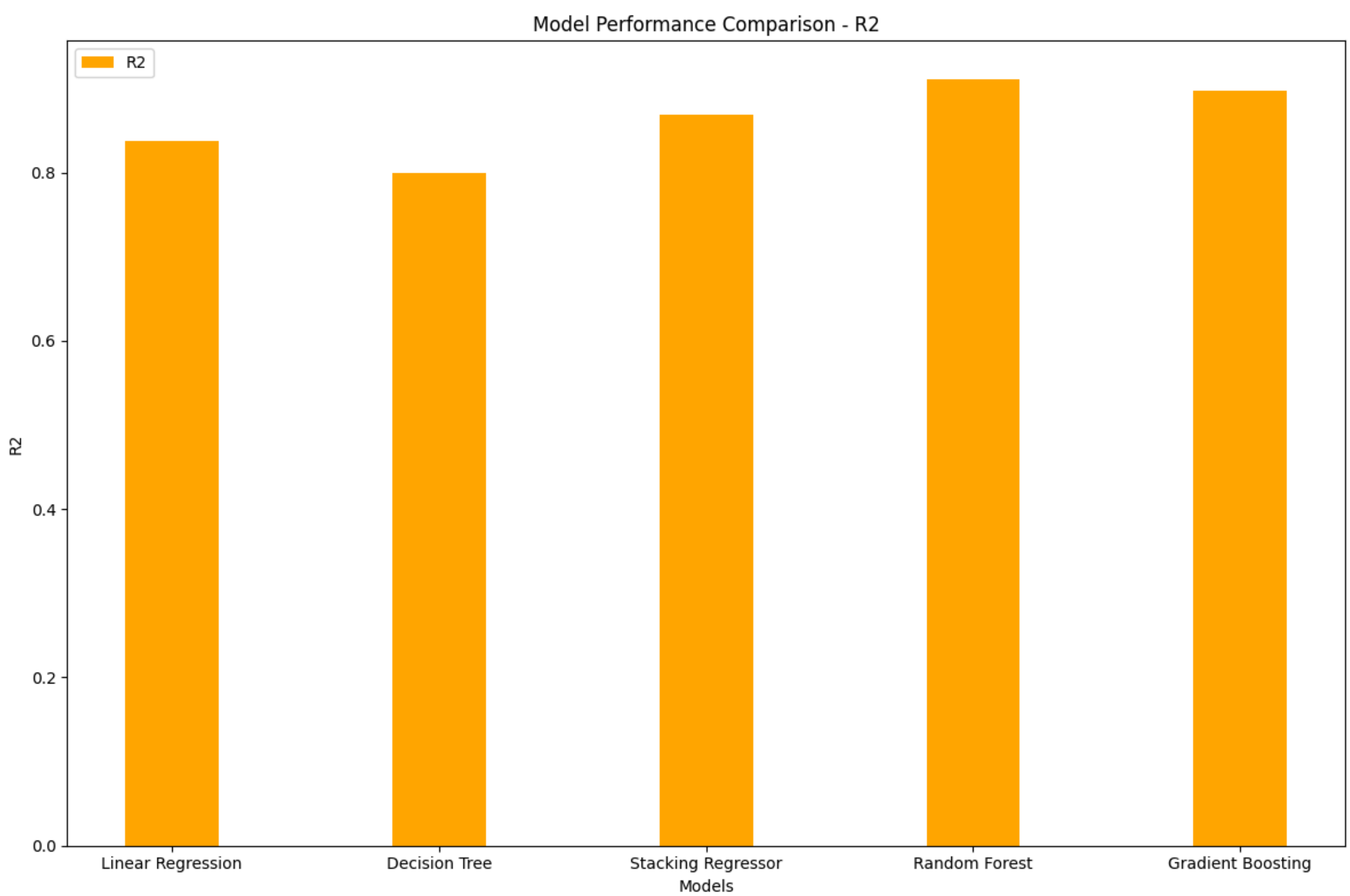
We can see that Random Forest seems to be the best model among these, boasting the lowest MSE (0.1231) and the highest R2 (0.9110), suggesting a strong relationship of 91% between the predicted happiness score and actual happiness score.

Gradient Boosting is the second best model here, with second lowest MSE (0.1415) and second highest R2 (0.8976).

I've also plotted the graph below.

In [19]:
```
x = np.arange(5)
width = 0.35

# Plot MSE scores
fig, ax1 = plt.subplots(figsize=(12, 8))
bars1 = ax1.bar(x, mse_scores, width, label='MSE')
ax1.set_xlabel('Models')
ax1.set_ylabel('MSE')
ax1.set_title('Model Performance Comparison - MSE')
ax1.set_xticks(x)
ax1.set_xticklabels(model_names)
ax1.legend(loc='upper left')

fig.tight_layout()

# Plot R2 scores
fig, ax2 = plt.subplots(figsize=(12, 8))
bars2 = ax2.bar(x, r2_scores, width, label='R2', color='orange')
ax2.set_xlabel('Models')
ax2.set_ylabel('R2')
ax2.set_title('Model Performance Comparison - R2')
ax2.set_xticks(x)
ax2.set_xticklabels(model_names)
ax2.legend(loc='upper left')

fig.tight_layout()

plt.show()
```





Model Optimization

Now we are going to optimize the Random Forest model and Gradient Boosting model.

I will use grid search to find the best hyperparameter for each models.

In [20]:
```
param_grids = {
    'Decision Tree': {'max_depth': [5, 10, 20], 'min_samples_split': [2, 10, 20], 'min_s
    'Random Forest': {'n_estimators': [50, 100, 200], 'max_depth': [10, 20],
                      'min_samples_split': [2, 10], 'min_samples_leaf': [1, 10]},
    'Gradient Boosting': {'n_estimators': [50, 100, 200], 'learning_rate': [0.05, 0.1, 0
                          'min_samples_split': [2, 10], 'min_samples_leaf': [1, 10]}
}
grid_models = {
    'Decision Tree': dt,
    'Random Forest': rf,
    'Gradient Boosting': gb,
}

best_models = {}
for model_name, model in grid_models.items():
    grid_search = GridSearchCV(model, param_grids[model_name], cv=3, scoring='neg_mean_s
    grid_search.fit(X_train, y_train)
    best_models[model_name] = grid_search.best_estimator_
    print(f"Best parameters for {model_name}:", grid_search.best_params_)
    print(f"Best score for {model_name}:", -grid_search.best_score_)
```

```
Best parameters for Decision Tree: {'max_depth': 10, 'min_samples_leaf': 10, 'min_samples
_split': 2}
Best score for Decision Tree: 0.21924034686470115
Best parameters for Random Forest: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_
split': 2, 'n_estimators': 200}
Best score for Random Forest: 0.13169374278240822
Best parameters for Gradient Boosting: {'learning_rate': 0.1, 'max_depth': 5, 'min_sample
s_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
Best score for Gradient Boosting: 0.13187137091564147
```

In [21]:
```
rf_2 = RandomForestRegressor(max_depth=20, min_samples_leaf=1, min_samples_split=2, n_es
gb_2 = GradientBoostingRegressor(learning_rate=0.1, max_depth=5, min_samples_leaf=1, min

optimized_models = {
    'Random Forest (Optimized)': rf_2,
    'Gradient Boosting (Optimized)': gb_2,
}

for name, model in optimized_models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'{name} - MAE: {mae:.4f}, MSE: {mse:.4f}, R2: {r2:.4f}')
```

```
Random Forest (Optimized) - MAE: 0.2573, MSE: 0.1234, R2: 0.9107
Gradient Boosting (Optimized) - MAE: 0.2578, MSE: 0.1220, R2: 0.9118
```

After optimization using grid search, Gradient Boosing (Optimized) now has the lowest MSE (0.1220) and highest R2 (0.9118), taking the crown of the best model trained here.

Conclusion

In this project, we aimed to predict the Life Ladder score (a measure of happiness) using socio-economic factors from the World Happiness Report 2018 data. We followed a systematic approach involving data preprocessing, feature engineering, model training, evaluation, and optimization.

Key Steps and Findings:

1. **Data Preparation:**
   - Handled missing values by replacing them with country-specific averages.
   - Dropped irrelevant columns and standardized features.
2. **Model Training and Evaluation:**
   - Trained Linear Regression, Decision Tree, Random Forest, Gradient Boosting, and Stacking Regressors.
   - Evaluated models using MAE, MSE, and R2 metrics.
3. **Model Optimization:**
   - Performed hyperparameter tuning using Grid Search for Random Forest and Gradient Boosting.
   - The optimized Gradient Boosting model achieved the best performance with an MSE of 0.1220 and an R2 of 0.9118.

In the end, the optimized Gradient Boosting Regressor was my best model. This model accurately predicts the Life Ladder score, highlighting the effectiveness of ensemble methods and hyperparameter tuning in improving predictive performance.

In [ ]:
```

```