



The World's Happiness in Focus

Final Project - June 2021

Project Overview

This project aims to analyze the global state of ‘happiness’ in the world, from 2006 to 2020, from people in over 150 countries, using the United Nations Happiness Index dataset.



Data Story

Show the correlation between measurements of well-being and create a model to identify which one is essential to discovering the ‘key to happiness’.



Data Source

The raw dataset files were obtained from the Kaggle website as csv files.



Target Audience

Students and research professionals.

How It Was Done

01

• ETL/Web Design

02

• ML Model/
Database Structure

03

• Python "Flask" App

04

• Heroku
Deployment

Data Processing

After establishing our data sources and dependencies, the ETL was implemented.

- null values/duplicates were removed
- columns renamed accordingly
- Groupby/sort/filter data accordingly

RAW DATA

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1949 entries, 0 to 1948
Data columns (total 11 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Country name                          1949 non-null   object
1   year                                  1949 non-null   int64
2   Life Ladder                           1949 non-null   float64
3   Log GDP per capita                     1913 non-null   float64
4   Social support                         1936 non-null   float64
5   Healthy life expectancy at birth       1894 non-null   float64
6   Freedom to make life choices            1917 non-null   float64
7   Generosity                             1860 non-null   float64
8   Perceptions of corruption              1839 non-null   float64
9   Positive affect                        1927 non-null   float64
10  Negative affect                        1933 non-null   float64
dtypes: float64(9), int64(1), object(1)
memory usage: 167.6+ KB
None
```

TRANSFORMED DATA

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1708 entries, 0 to 1948
Data columns (total 11 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Country name                          1708 non-null   object
1   year                                  1708 non-null   int64
2   Life Ladder                           1708 non-null   float64
3   Log GDP per capita                     1708 non-null   float64
4   Social support                         1708 non-null   float64
5   Healthy life expectancy at birth       1708 non-null   float64
6   Freedom to make life choices            1708 non-null   float64
7   Generosity                             1708 non-null   float64
8   Perceptions of corruption              1708 non-null   float64
9   Positive affect                        1708 non-null   float64
10  Negative affect                        1708 non-null   float64
dtypes: float64(9), int64(1), object(1)
memory usage: 160.1+ KB
None
```

ML Model

Scikit-learn library was used to build our data model.

- LinearRegression and Regression Trees (rt) was chosen to work with our numeric target variables.
- train_test_split was used to create training and testing data (80% train , 20% test).
- Fitted the model to the training data.
- Tested the model to the unseen test data.
- The model was used to make predictions for 2021 data.

VARIABLES

```
[7] X = happinessByYears_df[['GDP_per_capita','Life_expectancy']]
y = happinessByYears_df['Life_ladder'].values.reshape(-1, 1)
print(X.shape, y.shape)

(1708, 2) (1708, 1)
```

MODEL

```
[12] # Create the model using LinearRegression
from sklearn.linear_model import LinearRegression
model = LinearRegression()

# Fit the model to the training data and calculate the
model.fit(X_train, y_train)
training_score = model.score(X_train, y_train)
testing_score = model.score(X_test, y_test)

print(f"Training Score: {training_score}")
print(f"Testing Score: {testing_score}")

Training Score: 0.6423601987234213
Testing Score: 0.6718241224793313
```

```
# Fit the model to the training data and calculate the
model.fit(X_train, y_train)
training_score = model.score(X_train, y_train)
testing_score = model.score(X_test, y_test)
```

PREDICTIONS

	Prediction	Actual
0	7.159857	7.842
1	7.159857	7.620
2	7.159857	7.571
3	7.159857	7.554
4	7.159857	7.464
...
144	4.357920	3.512
145	4.357920	3.467
146	5.368271	3.415
147	4.357920	3.145
148	4.357920	2.523

149 rows × 2 columns

Database

The transformed main dataframes and predictions dataframe were successfully loaded to the PostgreSQL database using sqlalchemy

```
[11] ▶ ML  
happinessByYears_df.to_sql(name='happinessoveryears', con=engine, if_exists='append', index=False)  
statistics.to_sql(name='statistics', con=engine, if_exists='append', index=False)  
gbMainData.to_sql(name='gbmaindata', con=engine, if_exists='append', index=False)  
bottom.to_sql(name='bottom', con=engine, if_exists='append', index=False)  
top.to_sql(name='top', con=engine, if_exists='append', index=False)
```

DATA

Datastores > postgresql-slippery-31945

SERVICE heroku-postgresqlPLAN hobby-devBILLING APP flask-maf

OverviewDurabilitySettingsDataclips

HEALTH

Available

PRIMARY YesVERSION 13.2CREATED 5 days agoMAINTENANCE UnsupportedROLLBACK Unsupported

UTILIZATION

1 of 20

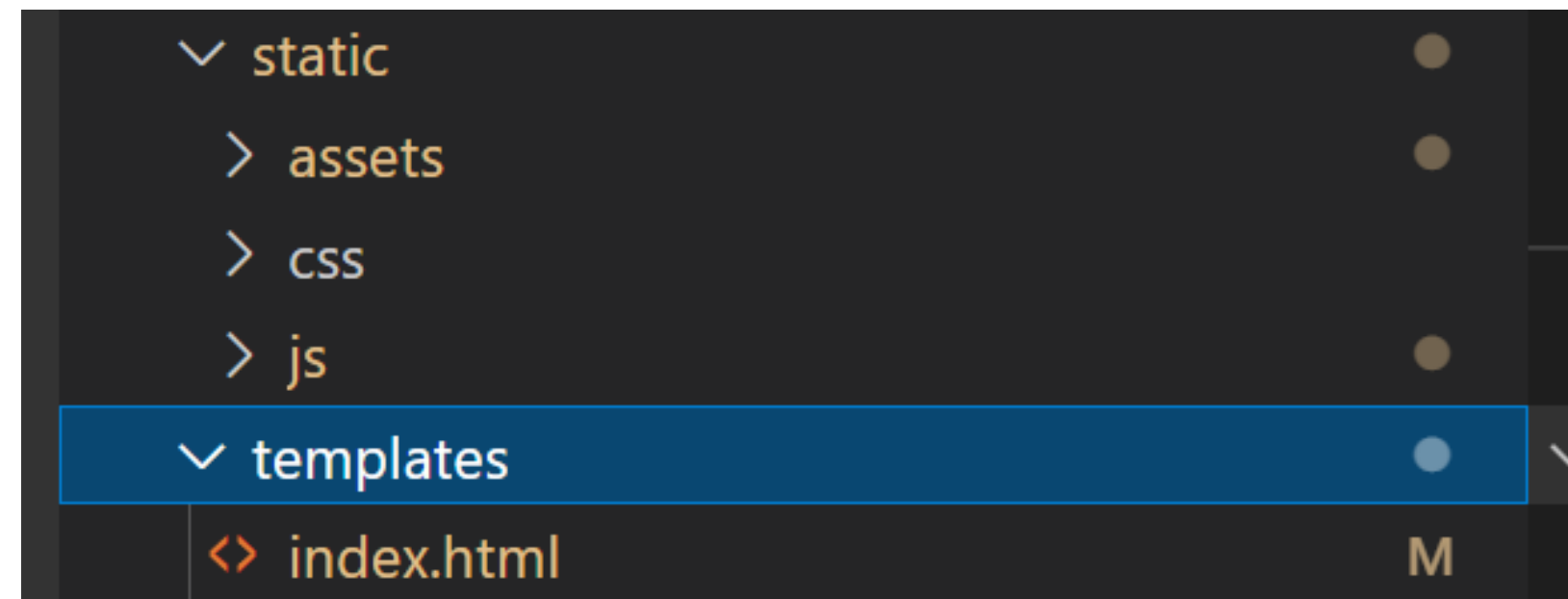
11,263 of 10,000

9.4 MB

4

Web Design

JavaScript, HTML, CSS from a Bootstrap template were used for this section. Changes were made accordingly.



Flask

Connection to the database was done via flask API Calls.
API routes for each table were created to get the data in json format for the visualizations.

```
project4 > main.py > ...
64 #     parsed = json.loads(result)
65 #     gbMainData_df_json = json.dumps(parsed, skipkeys = True, allow_nan = True, indent = 6)
66 #     return gbMainData_df_json
67
68 @app.route("/api/bottom")
69 def bottom():
70     result = bottom_df.to_json(orient="records")
71     parsed = json.loads(result)
72     bottom_df_json = json.dumps(parsed, skipkeys = True, allow_nan = True, indent = 6)
73     return bottom_df_json
74
75 @app.route("/api/top")
76 def top():
77     result = top_df.to_json(orient="records")
78     parsed = json.loads(result)
79     top_df_json = json.dumps(parsed, skipkeys = True, allow_nan = True, indent = 6)
80     return top_df_json
81
82 @app.route("/api/happiness2021")
83 def happiness2021():
84     result = happiness2021_df.to_json(orient="records")
85     parsed = json.loads(result)
86     happiness2021_df_json = json.dumps(parsed, skipkeys = True, allow_nan = True, indent = 6)
87     return happiness2021_df_json
```


Visualizations

async function was used in our JavaScript visualization file to connect to the database via API call.

For Loop was used to get each individual variables from the json objects for the traces.

D3 and Plotly were used for the charts/maps

```
1
2  const api_url_main = '/api/main'
3  |
4  async function getData_main(){
5  const response_main = await fetch(api_url_main)
6  const data_main = await response_main.json();
7  console.log(data_main)
8
9  let x1 = []
10 let y1 = []
11 let Country = []
12 let Corruption = []
13 let Freedom =[]
14 let GDP_per_capita =[]
15 let Generosity =[]
16 let Life_expectancy =[]
17 let Social_support = []
18
19 for (var i=0; i< data_main.length; i++){
20 x1.push(data_main[i]['year'])
21 y1.push(data_main[i]['Life_ladder'])
22 Corruption.push(data_main[i]['Corruption'])
23 Freedom.push(data_main[i]['Freedom'])
24 GDP_per_capita.push(data_main[i]['GDP_per_capita'])
25 Generosity.push(data_main[i]['Generosity'])
26 Country.push(data_main[i]['Country'])
27 Social_support.push(data_main[i]['Social_support'])
28 Life_expectancy.push(data_main[i]['Life_expectancy'])}
29
30 // add choropleth map (avg Life_ladder/ years & countries)
31 var data = [{
```

Heroku Deployment



Jump to Favorites, Apps, Pipelines, Spaces...

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more](#).

Choose a branch to deploy

main

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more](#).

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub ✓

Build main 76142da3 ✓

Release phase ✓

Deploy to Heroku ✓

Your app was successfully deployed.

View

Questions ?

Thank You

