# Asset Wealth Forecast for African Regions based on Remote Sensing Data

***2022***

**Pia Störmer**

# TABLE OF CONTENTS

CHAPTER

ONE

# HARDWARE AND SOFTWARE REQUIREMENTS

## This code was tested on a system with the following specifications:

- operating system: 20.04.1-Ubuntu SMP
- CPU: AMD EPYC 7443P 24-Core
- GPU: 1x NVIDIA RTX A6000

## Software Requirements:

- Python Version: 3.8.8
- Tensorflow Version: 2.8
- Keras: 2.8

*Further Python Package requirements are listed in the requirements.txt.*

CHAPTER

TWO

# DATA ACQUISITION & PREPROCESSING

1. Calculate Asset Wealth: `/src/dhs_preparation.py`.
2. Set Parameters for Satellite Data Retrieval inside `/src/config.py`.
3. Export satellite images from Google Earth Engine:
    1. `/src/ee_sentinel.py` for Sentinel-2 Data
    2. `/src/ee_viirs.py` for VIIRS Data
4. Move Files to corresponding Preprocessing Folders by using `/notebooks/split_geotiffs_for_preprocessing.ipynb`.
5. Preprocess GeoTIFFs: `/src/preprocess_geodata.py`.

CHAPTER

THREE

# MODEL TRAINING

1. Set Parameters for Model Training inside `/src/config.py`.
2. Run `/src/train_directly.py` and login to Weights & Bias to track Model Training and Evaluation.

CHAPTER

# FOUR

# NOTEBOOKS

1. Use `/notebooks/asset_wealth_analysis.ipynb` to analyze the calculated Asset Wealth.
2. Use `/notebooks/asset_wealth_prognosis.ipynb` to analyze test results and predict Asset Wealth for Mozambique (2016, 2017,2019, 2020 and 2021).

## 4.1 Python Scripts

### 4.1.1 src.config

### 4.1.2 src.data_utils

`src.data_utils.`**`calc_mean`**(*img_dir: str*, *img_list: list*, *input_height: int*, *input_width: int*, *clipping_values: list*, *channels: list*)

Calculate mean Pixel Values per Channel over all Input Images :param img_dir: Path to Image Data :type img_dir: str :param input_height: Pixel Height of Input :type input_height: int :param input_width: Pixel Width of Input :type input_width: int :param clipping_values: Interval of Min and Max Values for Clipping :type clipping_values: list :param channels: Channels to use; [] if all Channels are to be used :type channels: list

Returns (np.array): Means of pixel values per channel

`src.data_utils.`**`calc_std`**(*means*, *img_dir: str*, *img_list: list*, *input_height: str*, *input_width: str*, *clipping_values: list*, *channels: list*)

Calculate Standard Deviation Values per Channel over all Input Images :param means: Result of calc_mean: Mean of Pixel Values for each Channel :type means: np.array :param img_dir: Path to Image Data :type img_dir: str :param input_height: Pixel Height of Input :type input_height: int :param input_width: Pixel Width of Input :type input_width: int :param clipping_values: Interval of Min and Max Values for Clipping :type clipping_values: list :param channels: Channels to use; [] if all Channels are to be used :type channels: list

Returns (np.array): Standard deviation of Pixel Values per Channel

`src.data_utils.`**`combine_wealth_dfs`**(*wealth_csv_path: str*)

Combines all label csv files to one. :param wealth_csv_path: Path to Cluster CSV Files :type wealth_csv_path: str

**Returns** Pandas Dataframe containing all Clusters

**Return type** complete_wealth_df (pd.DataFrame)

`src.data_utils.`**`create_splits`**(*img_dir: str*, *pre2015_path: str*, *wealth_path: str*, *urban_rural: str*, *subset=False*)

Create train/val and testsplit for Cross Validation. :param img_dir: Path to Image Directory :type img_dir: str :param pre2015_path: Path to Test Images with corresponding Label dated older than 2015 :type pre2015_path: str :param wealth_path: Path to Label CSV Files :type wealth_path: str :param urban_rural: One of [‘u’,’r’,’ur’] to choose whether to use only Urban/Rural Clusters or all Data :type urban_rural: str :param subset: Whether or not to use a subset (for testing) :type subset: bool

**Returns**

List containing Filenames for Train and Validation Split X_test (list): List containing Filenames for Test Split y_train_val (np.ndarray): Numpy Array containing Asset Wealth (Label Data) for Train and Validation Split y_test (np.ndarray): Numpy Array containing Asset Wealth (Label Data) for Test Split

If pre2015_path is also returns: X_test_pre2015 (list): List containing Filenames for Test Dataset with corresponding Label dated older than 2015 y_test_pre2015 (np.ndarray):Numpy Array containing Asset Wealth (Label Data) for Test Dataset (dated older than 2015)

**Return type** X_train_val (list)

`src.data_utils.`**`generator`**(*img_dir: str*, *X: list*, *y: numpy.ndarray*, *batch_size: int*, *input_height: int*, *input_width: int*, *channel_size: int*)

Data generator to generate Label and Feature Batches. :param img_dir: Path to Image Directory :type img_dir: str :param X: List containing Filenames of Split :type X: list :param y: Array containing Label Values of Split :type y: np.ndarray :param batch_size: Size of Training Batches :type batch_size: int :param input_height: Pixel Height of Input :type input_height: int :param input_width: Pixel Width of Input :type input_width: int :param channels: Number of Channels :type channels: int

**Returns** batch_x (np.ndarray): Feature Batch batch_y (np.ndarray): Label Batch

`src.data_utils.`**`get_img_coordinates`**(*img: str*)

Extract the Cluster Coordinates from a given Filename. :param img: Filename of Image :type img: str

**Returns** Latitude, Longitude

**Return type** str, str

`src.data_utils.`**`get_kurtosis`**(*wealth_df: pandas.core.frame.DataFrame*)

Calculate the Kurtosis for WEALTH_INDEX column of a Pandas DataFrame

**Parameters** **`wealth_df`** – Pandas Dataframe containing at least a Column ‘WEALTH_INDEX’

**Returns** Mean Asset Wealth of DataFrame

**Return type** float

`src.data_utils.`**`get_label_for_img`**(*wealth_df: pandas.core.frame.DataFrame*, *img_filename: str*)

Get Label Data for a Cluster based on the Filename. :param wealth_df: Pandas Dataframe containing all Clusters :type wealth_df: pd.DataFrame :param img_dir: Path to Image Directory

**Returns** Dataframe including the Asset Wealth Value

**Return type** wealth_sentinel_df

`src.data_utils.`**`get_mean`**(*wealth_df: pandas.core.frame.DataFrame*)

Calculate the Mean Value for WEALTH_INDEX column of a Pandas DataFrame

**Parameters** **`wealth_df`** – Pandas Dataframe containing at least a Column ‘WEALTH_INDEX’

**Returns** Mean Asset Wealth of DataFrame

**Return type** float

`src.data_utils.get_median(wealth_df: pandas.core.frame.DataFrame)`

Calculate the Median Value for WEALTH_INDEX column of a Pandas DataFrame

**Parameters** `wealth_df` – Pandas Dataframe containing at least a Column 'WEALTH_INDEX'

**Returns** Median Asset Wealth of DataFrame

**Return type** float

`src.data_utils.get_skew(wealth_df: pandas.core.frame.DataFrame)`

Calculate the Skewness for WEALTH_INDEX column of a Pandas DataFrame

**Parameters** `wealth_df` – Pandas Dataframe containing at least a Column 'WEALTH_INDEX'

**Returns** Mean Asset Wealth of DataFrame

**Return type** float

`src.data_utils.get_statistics(csv_path: str, timespan_a: list, countries: list, timespan_b=False, timespan_c=False)`

Creates a Dictionary that includes statistic values per country year and combined per timespan. The Dictionary has the following structure: statistics = { 'country_year': [], 'mean': [], 'median': [], 'std': [], 'var': [], 'skewness': [], 'kurtosis': [] } :param csv_path: Path to Cluster CSV Files :type csv_path: str :param timespan_a: Timespan in Years e.g. [2012,2013,2014] to include :type timespan_a: list :param countries: Countries to include :type countries: list :param timespan_b: Optional: Second Timespan in Years e.g. [2015] to include :type timespan_b: bool/list :param timespan_c: Optional: Third Timespan in Years e.g. [2016, 2017,2018,2019,2020] to include :type timespan_c: bool/list

**Returns** Dictionary including statistic values per country year and combined over timespan(s)

**Return type** statistics (dict)

`src.data_utils.get_std(wealth_df: pandas.core.frame.DataFrame)`

Calculate the Standard Deviation for WEALTH_INDEX column of a Pandas DataFrame

**Parameters** `wealth_df` – Pandas Dataframe containing at least a Column 'WEALTH_INDEX'

**Returns** Mean Asset Wealth of DataFrame

**Return type** float

`src.data_utils.get_ur_statistics(csv_path: str, timespan_a: list, countries: list, timespan_b=False, timespan_c=False)`

Creates a Dictionary that includes statistic per region type (urban/rural) per timespan. The Dictionary has the following keys: statistics = { 'year': [], 'ur': [], 'mean': [], 'median': [], 'std': [], 'var': [], 'skewness': [], 'kurtosis': [] } :param csv_path: Path to Cluster CSV Files :type csv_path: str :param timespan_a: Timespan in Years e.g. [2012,2013,2014] to include :type timespan_a: list :param countries: Countries to include :type countries: list :param timespan_b: Optional: Second Timespan in Years e.g. [2015] to include :type timespan_b: bool/list :param timespan_c: Optional: Third Timespan in Years e.g. [2016, 2017,2018,2019,2020] to include :type timespan_c: bool/list

**Returns** Dictionary including statistic values per region type (urban/rural) per timespan.

**Return type** statistics (dict)

`src.data_utils.get_var(wealth_df: pandas.core.frame.DataFrame)`

Calculate the Variance for WEALTH_INDEX column of a Pandas DataFrame

**Parameters** `wealth_df` – Pandas Dataframe containing at least a Column 'WEALTH_INDEX'

**Returns** Mean Asset Wealth of DataFrame

**Return type** float

`src.data_utils.`**`truncate`**(*f*, *n*)

Truncates a float f to n decimal places without rounding :param f: float value :param n: number of decimal places

### 4.1.3 src.dhs_preparation

**`class`** `src.dhs_preparation.`**`DHS_preparation`**(*floor_recode: dict*, *toilet_recode: dict*, *water_recode: dict*, *country_code_dict: dict*, *features: list*, *info: list*, *dhs_survey_path: str*, *wealth_path: str*, *shape_path: str*, *geo_wealth_path: str*, *sustainlab_group_file: str*)

Bases: `object`

**`create_wealth_geo_df`**(*shape_file: str*)

**`recode_and_format_dhs`**(*filename: str*)

**`split_sustainlab_clusters`**()

`src.dhs_preparation.`**`main`**()

### 4.1.4 src.ee_sentinel

`src.ee_sentinel.`**`bounding_box`**(*loc: ee.geometry.Geometry.Point*, *urban_rural: int*, *urban_radius: int*, *rural_radius: int*)

Function to get a square around point of interest Rural : 10 km Radius Urban : 2 km Radius :param loc: Geolocation of Cluster (from DHS Survey) :type loc: ee.Geometry.Point :param urban_rural: Binary Encoding for Region Type: 0 = urban, 1 = rural :type urban_rural: int :param urban_radius: Radius around Coordinates for Urban Regions in Meter :type urban_radius: int :param rural_radius: Radius around Coordinates for Rural Regions in Meter :type rural_radius: int

**Returns**

**bounding box around cluster coordinates** with a size of 10x10km for rural/ 2x2km for urban

**Return type** intermediate_box (ee.Geometry)

`src.ee_sentinel.`**`download_local`**(*survey_dir: str*)

Download Images from GoogleDrive Folder. :param survey_dir: Output Directory for Download :type survey_dir: str

`src.ee_sentinel.`**`get_image`**(*cluster: object*, *urban_radius: int*, *rural_radius: int*, *country_code: str*, *MAX_CLOUD_PROBABILITY: int*)

Extract Information about Cluster to get Sentinel2 Image for corresponding Year and Coordinates. :param cluster: Information about the Cluster (Cluster number, Coordinates, Survey Name, etc.) :type cluster: DictReader object :param survey_name: Name of the Survey (COUNTRY_YEAR) :type survey_name: str :param urban_radius: Radius around Coordinates for Urban Regions in Meter :type urban_radius: int :param rural_radius: Radius around Coordinates for Rural Regions in Meter :type rural_radius: int :param country_code: ISO Code for Survey Country (COUNTRY) :type country_code: str :param MAX_CLOUD_PROBABILITY: % :type MAX_CLOUD_PROBABILITY: int

**Returns** Latitude_Longitude-begin-end-country_r/u_sidelength Koordinaten: 4 Nachkommastellen Datumsformat: YYYYMMDD Land: Offizielle 3 Buchstaben Abkürzung (ISO) Rural und Urban: durch u bzw r Side length: Seitenlänge (Größe) der Kachel in km mit einer Nachkommastelle

**Return type** Requests Image from Earth Engine. Files are named by the following pattern `src.ee_sentinel.`**`get_survey_images`**(*file_dir: str*, *survey_name: str*, *urban_radius: int*, *rural_radius: int*, *MAX_CLOUD_PROBABILITY: int*)

Get Sentinel2 Image for each Cluster and download from GoogleDrive. :param file_dir: Path to DHS Survey CSV File :type file_dir: str :param survey_name: Name of the Survey (COUNTRY_YEAR) :type survey_name: str :param urban_radius: Radius around Coordinates for Urban Regions in Meter :type urban_radius: int :param rural_radius: Radius around Coordinates for Rural Regions in Meter :type rural_radius: int :param MAX_CLOUD_PROBABILITY: % :type MAX_CLOUD_PROBABILITY: int

`src.ee_sentinel.`**`maskClouds`**(*img: ee.image.Image*, *MAX_CLOUD_PROBABILITY: int*)

Masking of clouds :param img: Sentinel 2 Image retrieved from ee :type img: ee.Image :param MAX_CLOUD_PROBABILITY: % :type MAX_CLOUD_PROBABILITY: int

**Returns** CloudMasked EarthEngine Image

**Return type** ee.Image

`src.ee_sentinel.`**`sentinel_img_survey`**(*img_dir: str*, *csv_dir: str*, *sentinel_done: str*, *urban_radius: int*, *rural_radius: int*, *MAX_CLOUD_PROBABILITY: int*)

Iterate over Survey CSVs and get Sentinel2 Images for each Cluster. :param img_dir: Path to Directory where the Sentinel Images are stored :type img_dir: str :param csv_dir: Path to Directory where DHS CSV Files are stored :type csv_dir: str :param sentinel_done: Filepath for File to document for which Surveys were are already completed :type sentinel_done: str :param urban_radius: Radius around Coordinates for Urban Regions in Meter :type urban_radius: int :param rural_radius: Radius around Coordinates for Rural Regions in Meter :type rural_radius: int :param MAX_CLOUD_PROBABILITY: % :type MAX_CLOUD_PROBABILITY: int

### 4.1.5 src.ee_viirs

`src.ee_viirs.`**`bounding_box`**(*loc*, *urban_rural*, *urban_radius*, *rural_radius*)

Function to get a square around point of interest Rural : 10 km Radius Urban : 2 km Radius :param loc: Geolocation of Cluster (from DHS Survey) :type loc: ee.Geometry.Point :param urban_rural: Binary Encoding for Region Type: 0 = urban, 1 = rural :type urban_rural: int :param urban_radius: Radius around Coordinates for Urban Regions in Meter :type urban_radius: int :param rural_radius: Radius around Coordinates for Rural Regions in Meter :type rural_radius: int

**Returns**

**bounding box around cluster coordinates** with a size of 10x10km for rural/ 2x2km for urban

**Return type** intermediate_box (ee.Geometry)

`src.ee_viirs.`**`download_local`**(*survey_dir*)

Download Images from GoogleDrive Folder. :param survey_dir: Output Directory for Download :type survey_dir: str

`src.ee_viirs.`**`get_image`**(*cluster*, *survey_name*, *urban_radius*, *rural_radius*)

Extract Information about Cluster to get VIIRS Image for corresponding Year and Coordinates. :param cluster: Information about the Cluster (Cluster number, Coordinates, Survey Name, etc.) :type cluster: DictReader object :param survey_name: Name of the Survey (COUNTRY_YEAR) :type survey_name: str :param urban_radius: Radius around Coordinates for Urban Regions in Meter :type urban_radius: int :param rural_radius: Radius around Coordinates for Rural Regions in Meter :type rural_radius: int :param country_code: ISO Code for Survey Country (COUNTRY) :type country_code: str :param MAX_CLOUD_PROBABILITY: % :type MAX_CLOUD_PROBABILITY: int

**Returns** Latitude_Longitude-begin-end-country_r/u_sidelength Koordinaten: 4 Nachkommastellen Datumsformat: YYYYMMDD Land: Offizielle 3 Buchstaben Abkürzung (ISO) Rural und Urban: durch u bzw r Side length: Seitenlänge (Größe) der Kachel in km mit einer Nachkommastelle

**Return type** Requests Image from Earth Engine. Files are named by the following pattern

`src.ee_viirs.get_survey_images`(*file_dir*, *survey_name*, *urban_radius*, *rural_radius*)

Get VIIRS Image for each Cluster and download from GoogleDrive. :param file_dir: Path to DHS Survey CSV File :type file_dir: str :param survey_name: Name of the Survey (COUNTRY_YEAR) :type survey_name: str :param urban_radius: Radius around Coordinates for Urban Regions in Meter :type urban_radius: int :param rural_radius: Radius around Coordinates for Rural Regions in Meter :type rural_radius: int

`src.ee_viirs.viirs_img_survey`(*img_dir*, *csv_dir*, *viirs_done*, *urban_radius*, *rural_radius*)

Iterate over Survey CSVs and get VIIRS Images for each Cluster. :param img_dir: Path to Directory where the VIIRS Images are stored :type img_dir: str :param csv_dir: Path to Directory where DHS CSV Files are stored :type csv_dir: str :param viirs_done: Filepath for File to document for which Surveys were are already completed :type viirs_done: str :param urban_radius: Radius around Coordinates for Urban Regions in Meter :type urban_radius: int :param rural_radius: Radius around Coordinates for Rural Regions in Meter :type rural_radius: int

### 4.1.6 src.preprocess_geodata

`src.preprocess_geodata.main`(*img_path: str*, *ur: str*, *year: str*, *input_height: str*, *input_width: str*, *clipping_values: list*, *channels: list*, *add_img_path=False*, *standardize=False*)

**Parameters**

- **img_path** (*str*) – Path to Image Data
- **ur** (*str*) – 'u' for urban, 'r' for rural
- **year** (*str*) – timespan (2012_2014 / 2016_2020) or all data
- **input_height** (*int*) – Desired Input Height
- **input_width** (*int*) – Desired Input Width
- **clipping_values** (*list*) – Interval of Min and Max Values for Clipping
- **channels** (*list*) – List of Channels to use. [] to use all channels.
- **add_img_path** (*bool/str*) – Optional: Path to Image Data to add (eg. for combining Sentinel2 and VIIRS)
- **standardize** – Optional: Whether or not to standardize Image Data (e.g. standardization is not needed when already normalized Sentinel2 and VIIRS data are merged)

`src.preprocess_geodata.slice_to_input_size`(*array: numpy.ndarray*, *input_height: int*, *input_width: int*)

**Parameters**

- **array** (*np.ndarray*) – Numpy Array containing Image Data
- **input_height** (*int*) – Uniform Image Height to slice to
- **input_width** (*int*) – Uniform Image Width to slice to

**Returns** Numpy Array containing Image Data in shape of Input Height/Width and Bandwidth

**Return type** array (np.array)

`src.preprocess_geodata.standardize_resize`(*img: str*, *img_path: str*, *input_height: str*, *input_width: str*, *clipping_values: list*, *means=False*, *stds=False*, *add_img_path=False*, *standardize=False*)

Standardize and Resize GeoTIFFs. Standardization is performed per Band using Standard Scaler. Resizing is performed by slicing to the Center of the Image in Shape of provided Input Size. For VIIRS Images, the Band is tripled to fit RGB Input Shape of common CNNs. Standardized and Resized Images are stored in a new GeoTIFF. :param img: Filename of Image to Normalize and Resize :type img: str :param img_path: Path to Image Data :type img_path: str :param input_height: Desired Input Height :type input_height: int :param input_width: Desired Input Width :type input_width: int :param clipping_values: Interval of Min and Max Values for Clipping :type clipping_values: list :param means: Optional: Result of calc_mean: Mean of Pixel Values for each Channel :type means: bool/np.ndarray :param stds: Optional: Result of calc_mean: Standard Deviation of Pixel Values for each Channel :type stds: bool/np.ndarray :param add_img_path: Optional: Path to Image Data to add (eg. for combining Sentinel2 and VIIRS) :type add_img_path: bool/str :param standardize: Optional: Whether or not to standardize Image Data (e.g. standardization is not needed when already normalized Sentinel2 and VIIRS data are merged) :type standardize: bool

### 4.1.7 src.rename_viirs

`src.rename_viirs.`**`get_center_coords`**`(`*img_path=<class 'str'>*`)`

Get Center Coordinates of a GeoTIFF :param img_path: Path to GeoTIFF :type img_path: str

**Returns** Center Longitude Value of Image lat (float): Center Latitude Value of Image

**Return type** long (float)

`src.rename_viirs.`**`main`**`(`*img_dir: str*`)`

Rename all VIIRS GeoTIFFS from DIS22 according to filename pattern: Latitude_Longitude-begin-end-country_r/u_sidelength Koordinaten: 4 Nachkommastellen Datumsformat: YYYYMMDD Land: Offizielle 3 Ziffern Abkürzung Rural und Urban: durch u bzw r Side length: Seitenlänge (Größe) der Kachel in km mit einer Nachkommastelle :param img_dir: Path to image data

Returns:

### 4.1.8 src.resnet50

**class** `src.resnet50.`**`ResNet50v2_hyperspectral`**`(`*img_w: int*, *img_h: int*, *channels: int*`)`

Bases: `object`

**`load_resnet50v2`**`()`

Returns a Resnet50v2 *keras.Model* instance fitted to Hyperspectral/RGB image input

### 4.1.9 src.train

`src.train.`**`main`**`(`*img_dir: str*, *csv_path: str*, *pre2015_path: str*, *model_name: str*, *k: int*, *input_height: int*, *input_width: int*, *img_source: str*, *urban_rural: str*, *channel_size: int*, *batch_size: int*, *epochs: int*, *subset: bool*`)`

Train a Model with the Parameters set in config.py. :param img_dir: Path to Image Data :type img_dir: str :param csv_path: Path to Cluster CSV Files :type csv_path: str :param pre2015_path: Path to Image Data older than 2015; if all Data is used for training this shoulde be False. :type pre2015_path: str :param model_name: One of [‘vgg19’, ‘resnet50’] to choose which Model is used :type model_name: str :param k: Number of Folds for Cross Validation :type k: int :param input_height: Pixel Height of input :type input_height: int :param input_width: Pixel Width of input :type input_width: int :param img_source: One of [‘s2’, ‘viirs’] to choose whether Sentinel-2, VIIRS (nightlight) or combined Data is used :type img_source: str :param urban_rural: On of [‘u’,’r’,’ur’] to choose whether only urban or only rural clusters are used :type urban_rural: str :param channels: Channels to use; [] to use all Channels :type channels: list :param channel_size: Number of Channels (3 for RGB (VIIRS), 13 for all Sentinel2 Channels, 14 for all Channels) !Nightlight channel is transformed to 3 channels for Model Compatibility :type channel_size: int :param batch_size: Size of Training Batches :type batch_size: int :param epochs: Number of Training Epochs :type epochs: int :param subset: Whether or not to use a Subset to test the Process :type subset: bool

### 4.1.10 src.vgg19

**class** src.vgg19.**VGG19_hyperspectral**(*img_w: int*, *img_h: int*, *channels: int*)

Bases: object

**load_vgg19**()

Returns: Create a Model Template of VGG19 with RGB or hyperspectral input shape (as defined in init)

## 4.2 Notebooks

### 4.2.1 Statistical Analysis of Asset Wealth

[1]:
```
import os
import glob
import sys

sys.path.append("..")

import pandas as pd

import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

from src.data_utils import get_mean
from src.data_utils import get_median
from src.data_utils import get_std
from src.data_utils import get_var
from src.data_utils import get_skew
from src.data_utils import get_kurtosis
from src.data_utils import get_ur_statistics
from src.data_utils import get_statistics
```

#### Compare urban and rural Regions over Time

[2]:
```
csv_path='/mnt/datadisk/data/surveys/asset/dhs_data/label_data/'
countries = ['Malawi', 'Kenya', 'Democratic Republic of Congo',  'Rwanda', 'Zambia',
↪'Uganda', 'Tanzania', 'Ethiopia', 'Mozambique', 'Zimbabwe']
```

[3]:
```
statistics_ur = get_ur_statistics(csv_path=csv_path, timespan_a=range(2012,2015),␣
↪countries=countries, timespan_b = range(2016,2021), timespan_c=range(2015,2016))
```

[4]:
```
statistics_ur_df = pd.DataFrame.from_dict(statistics_ur, orient='columns')
print(statistics_ur_df.shape)
statistics_ur_df.head(2)
```

```
(6, 8)
```

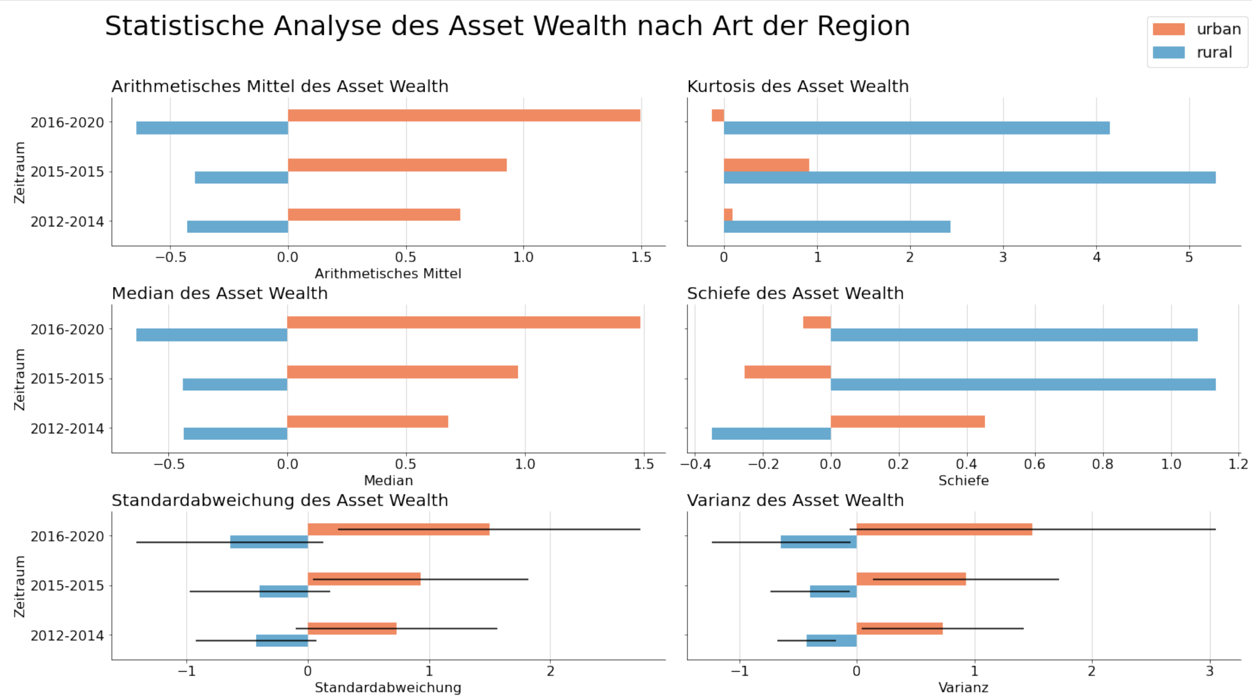
[4]:
```
        year     ur      mean    median       std       var  skewness  \
0  2012-2014  urban  0.732320  0.676740  0.831303  0.691065  0.453175
1  2012-2014  rural -0.426892 -0.435667  0.496990  0.246999 -0.350300

   kurtosis
0  0.085632
1  2.431044
```

[6]:
```
mosaic = """
    AB
    CD
    EF
    """
fig = plt.figure(constrained_layout=True)
axes = fig.subplot_mosaic(mosaic,sharey=True)

plt.rcParams["figure.figsize"]=(20,10)
for ax in [axes['A'],axes['B'],axes['C'],axes['D'],axes['E'],axes['F']]:
    ax.tick_params(axis='both', which='major', labelsize=16)
    ax.set_axisbelow(True)



fig.suptitle("Statistische Analyse des Asset Wealth nach Art der Region", x=.4, y=1.1,␣
↪fontsize=32)

axes['A'].set_title('Arithmetisches Mittel des Asset Wealth', fontsize=20, loc='left')
means = pd.pivot_table(statistics_ur_df,values="mean",index="year",columns="ur")
means.plot(
    kind='barh', ax=axes['A'], color=['#67a9cf', '#ef8a62'])
axes['A'].get_legend().remove()
axes['A'].set_yticks(statistics_ur_df.index[:3], means.index.unique())
axes['A'].set_xlabel('Arithmetisches Mittel', fontsize=16)
axes['A'].set_ylabel('Zeitraum', fontsize=16)
axes['A'].grid(axis='x',color='lightgrey')
axes['A'].spines['top'].set_visible(False)
axes['A'].spines['right'].set_visible(False)


axes['B'].set_title('Kurtosis des Asset Wealth', fontsize=20, loc='left')
pd.pivot_table(statistics_ur_df,values="kurtosis",index="year",columns="ur").plot(
    kind='barh', width=.5, ax=axes['B'], color=['#67a9cf', '#ef8a62'])
axes['B'].get_legend().remove()
axes['B'].set_ylabel('Zeitraum', fontsize=16)
axes['B'].grid(axis='x',color='lightgrey')
axes['B'].spines['top'].set_visible(False)
axes['B'].spines['right'].set_visible(False)
```

(continues on next page)

(continued from previous page)

```
axes['C'].set_title('Median des Asset Wealth', fontsize=20, loc='left')
pd.pivot_table(statistics_ur_df,values="median",index="year",columns="ur").plot(
    kind='barh', width=.5,ax=axes['C'], color=['#67a9cf', '#ef8a62'])
axes['C'].get_legend().remove()
axes['C'].set_xlabel('Median', fontsize=16)
axes['C'].set_ylabel('Zeitraum', fontsize=16)
axes['C'].grid(axis='x',color='lightgrey')
axes['C'].spines['top'].set_visible(False)
axes['C'].spines['right'].set_visible(False)

axes['D'].set_title('Schiefe des Asset Wealth', fontsize=20, loc='left')
pd.pivot_table(statistics_ur_df,values="skewness",index="year",columns="ur").plot(
    kind='barh', width=.5, ax=axes['D'], color=['#67a9cf', '#ef8a62'])
axes['D'].get_legend().remove()
axes['D'].set_xlabel('Schiefe', fontsize=16)
axes['D'].set_ylabel('Zeitraum', fontsize=16)
axes['D'].grid(axis='x',color='lightgrey')
axes['D'].spines['top'].set_visible(False)
axes['D'].spines['right'].set_visible(False)


axes['E'].set_title('Standardabweichung des Asset Wealth', fontsize=20, loc='left')
pd.pivot_table(statistics_ur_df,values=["mean",'std'],index="year",columns="ur").plot(
    kind='barh', y='mean', width=.5, ax=axes['E'], xerr='std', color=['#67a9cf', '#ef8a62
↪'])
axes['E'].get_legend().remove()
axes['E'].set_xlabel('Standardabweichung', fontsize=16)
axes['E'].set_ylabel('Zeitraum', fontsize=16)
axes['E'].grid(axis='x',color='lightgrey')
axes['E'].spines['top'].set_visible(False)
axes['E'].spines['right'].set_visible(False)


axes['F'].set_title('Varianz des Asset Wealth', fontsize=20, loc='left')
pd.pivot_table(statistics_ur_df,values=["mean",'var'],index="year",columns="ur").plot(
    kind='barh', y='mean', width=.5, ax=axes['F'], xerr='var', color=['#67a9cf', '#ef8a62
↪'])
axes['F'].get_legend().remove()
axes['F'].set_xlabel('Varianz', fontsize=16)
axes['F'].set_ylabel('Zeitraum', fontsize=16)
axes['F'].grid(axis='x',color='lightgrey')
axes['F'].spines['top'].set_visible(False)
axes['F'].spines['right'].set_visible(False)


patch_urban = mpatches.Patch(color='#ef8a62', label='urban')
patch_rural = mpatches.Patch(color='#67a9cf', label='rural')
fig.legend(handles=[patch_urban, patch_rural], fontsize=18, loc='upper right')
# plt.savefig('./asset_wealth_statistic_analysis.png', dpi=300, bbox_inches='tight', pad_
↪inches = 0)
plt.show()
```




[7]:
```
statistics_ur_df['year'] = statistics_ur_df.year.apply(lambda x:x.replace('_','-'))
statistics_ur_df['year'] = statistics_ur_df.year.apply(lambda x: x.replace('2015-2015',
↪'2015'))
statistics_ur_df.dropna().sort_values(by=['year'])
```

[7]:
```
        year     ur      mean    median       std       var  skewness  \
0  2012-2014  urban  0.732320  0.676740  0.831303  0.691065  0.453175
1  2012-2014  rural -0.426892 -0.435667  0.496990  0.246999 -0.350300
4       2015  urban  0.929695  0.971463  0.888563  0.789544 -0.252657
5       2015  rural -0.394422 -0.440112  0.580919  0.337467  1.132054
2  2016-2020  urban  1.497189  1.486528  1.246898  1.554755 -0.082131
3  2016-2020  rural -0.642661 -0.633979  0.768087  0.589958  1.080916

   kurtosis
0  0.085632
1  2.431044
4  0.912855
5  5.286251
2 -0.134592
3  4.150475
```

### Compare Statistics per Survey

[8]:
```
statistics = get_statistics(csv_path=csv_path, timespan_a=range(2012,2015),␣
↪countries=countries, timespan_b = range(2016,2021), timespan_c=range(2015,2016))
```

[9]:
```
cc_mapping = {
    'CD': 'COD',
    'ET': 'ETH',
    'KE': 'KEN',
    'MW': 'MWI',
    'MZ': 'MOZ',
    'RW': 'RWA',
    'TZ': 'TZA',
    'UG': 'UGA',
    'ZM': 'ZMB',
    'ZW': 'ZWE'
}
```

[10]:
```
statistics_df = pd.DataFrame.from_dict(statistics, orient='columns')
statistics_df = statistics_df.sort_values(by=['country_year']).reset_index(drop=True)
statistics_df['country_year'] = statistics_df.country_year.apply(lambda x: x.replace(x[:
↪2], cc_mapping[x[:2]]).replace('_', ' ') if not x.startswith('kombiniert') else x.
↪replace('kombiniert_', 'Kombiniert ').replace('_','-'))
statistics_df.loc[statistics_df.country_year=='Kombiniert 2016-2020', 'country_year'] =
↪'2016-2020'
statistics_df.loc[statistics_df.country_year=='Kombiniert 2012-2014', 'country_year'] =
↪'2012-2014'
statistics_df.loc[statistics_df.country_year=='Kombiniert 2015-2015', 'country_year'] =
↪'2015'
statistics_df.dropna()
```

[10]:

| | country_year | mean | median | std | var | skewness | kurtosis |
|---|---|---|---|---|---|---|---|
| 0 | COD 2014 | 0.038550 | -0.390554 | 0.958625 | 0.918962 | 2.005833 | 3.031307 |
| 1 | ETH 2016 | 0.012098 | -0.381761 | 0.929815 | 0.864555 | 0.923636 | -0.466208 |
| 2 | KEN 2014 | -0.002535 | -0.040012 | 0.846506 | 0.716573 | 0.031881 | 0.459236 |
| 3 | KEN 2015 | -0.001298 | -0.087786 | 0.775691 | 0.601697 | -0.001202 | 0.677301 |
| 4 | MWI 2014 | 0.003037 | -0.421321 | 0.818947 | 0.670674 | 1.185901 | 0.248446 |
| 5 | MWI 2015 | -0.066928 | -0.355052 | 0.678299 | 0.460089 | 1.820677 | 2.835156 |
| 6 | MWI 2016 | 0.213140 | -0.204208 | 0.868516 | 0.754319 | 1.053283 | 0.232369 |
| 7 | MOZ 2015 | 0.013473 | -0.375884 | 1.469633 | 2.159821 | 0.596114 | -0.635971 |
| 8 | MOZ 2018 | -0.016615 | -0.815378 | 1.832815 | 3.359211 | 0.549163 | -1.223835 |
| 9 | RWA 2014 | -0.151484 | -0.320774 | 0.550205 | 0.302726 | 2.661513 | 7.245296 |
| 10 | RWA 2015 | 0.095847 | -0.292310 | 0.877292 | 0.769642 | 1.595143 | 1.368602 |
| 11 | RWA 2020 | 0.002184 | -0.436307 | 1.324332 | 1.753856 | 1.179800 | 0.752779 |
| 12 | TZA 2015 | 0.145819 | -0.158977 | 0.889375 | 0.790989 | 0.608331 | -0.860105 |
| 13 | TZA 2016 | -0.448258 | -0.642479 | 0.553245 | 0.306080 | 1.637034 | 1.984532 |
| 14 | UGA 2014 | 0.096341 | -0.138693 | 0.836245 | 0.699306 | 1.024133 | 0.407601 |
| 15 | UGA 2015 | -0.147538 | -0.283633 | 0.857924 | 0.736034 | 1.497002 | 1.686292 |
| 16 | UGA 2019 | 0.019539 | -0.389447 | 1.398700 | 1.956362 | 1.135502 | 0.600526 |
| 17 | ZMB 2014 | -0.023153 | -0.377790 | 0.845999 | 0.715714 | 1.141197 | 0.337275 |
| 18 | ZMB 2018 | -0.006915 | -0.685355 | 1.797500 | 3.231007 | 0.795110 | -0.548185 |
| 19 | ZWE 2015 | 0.005259 | -0.367102 | 0.860724 | 0.740845 | 0.196717 | -1.641054 |
| 20 | 2012-2014 | -0.004615 | -0.194061 | 0.848463 | 0.719890 | 0.856188 | 1.222880 |

(continues on next page)

(continued from previous page)

```
21         2015  0.022697 -0.283227  0.926333  0.858093  0.841734  0.610966
22    2016-2020  0.001080 -0.415219  1.357643  1.843195  0.966412  0.410702
```

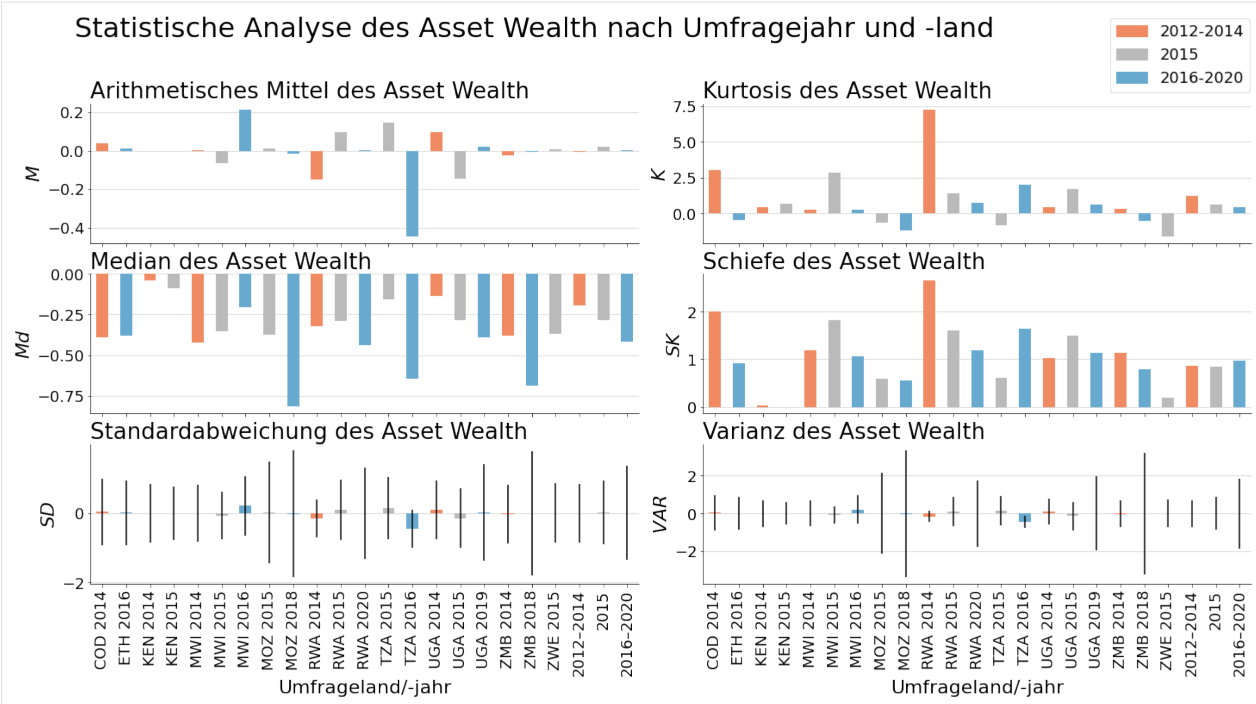
[11]:
```
c2012 = '#ef8a62'
c2015 = '#bababa'
c2016 = '#67a9cf'
colors = []
for cy in statistics_df.country_year.to_list():
    if any(str(year) in cy for year in range(2012,2015)):
        colors.append(c2012)
    elif '2015' in cy:
        colors.append(c2015)
    elif any(str(year) in cy for year in range(2016,2021)):
        colors.append(c2016)
```

[12]:
```
mosaic = """
    AB
    CD
    EF
    """
fig = plt.figure(constrained_layout=True)
axes = fig.subplot_mosaic(mosaic,sharex=True)

plt.rcParams["figure.figsize"]=(20,10)
for ax in [axes['A'],axes['B'],axes['C'],axes['D'],axes['E'],axes['F']]:
    ax.tick_params(axis='both', which='major', labelsize=18)
    ax.set_axisbelow(True)

fig.suptitle("Statistische Analyse des Asset Wealth nach Umfragejahr und -land", x=.42,␣
↪y=1.1, fontsize=32)

patch2012 = mpatches.Patch(color=c2012, label='2012-2014')
patch2015 = mpatches.Patch(color=c2015, label='2015')
patch2016 = mpatches.Patch(color=c2016, label='2016-2020')



axes['A'].set_title('Arithmetisches Mittel des Asset Wealth', fontsize=26, loc='left')
statistics_df['mean'].plot(kind='bar', ax=axes['A'], color=colors)#.set_
↪xticks(statistics_df.index[:-1], statistics_df.country_year.iloc[:-1])
axes['A'].set_ylabel('$\it{M}$', fontsize=22)
axes['A'].grid(axis='y',color='lightgrey')
axes['A'].spines['top'].set_visible(False)
axes['A'].spines['right'].set_visible(False)

axes['B'].set_title('Kurtosis des Asset Wealth', fontsize=26, loc='left')
statistics_df['kurtosis'].plot(kind='bar', width=.5, ax=axes['B'], color=colors)
axes['B'].set_ylabel('$\it{K}$', fontsize=22)
axes['B'].grid(axis='y',color='lightgrey')
axes['B'].spines['top'].set_visible(False)
axes['B'].spines['right'].set_visible(False)
```

(continues on next page)

(continued from previous page)

```
axes['C'].set_title('Median des Asset Wealth', fontsize=26, loc='left')
statistics_df['median'].plot(kind='bar', width=.5,ax=axes['C'], color=colors)#.set_
↪xticks(statistics_df.index[:-1], statistics_df.country_year.iloc[:-1])
axes['C'].set_xticks(statistics_df.index, statistics_df.country_year, fontsize=18)
axes['C'].set_ylabel('$\it{Md}$', fontsize=22)
axes['C'].set_xlabel('Umfrageland/-jahr', fontsize=22)
axes['C'].grid(axis='y',color='lightgrey')
axes['C'].spines['top'].set_visible(False)
axes['C'].spines['right'].set_visible(False)

axes['D'].set_title('Schiefe des Asset Wealth', fontsize=26, loc='left')
statistics_df['skewness'].plot(kind='bar', width=.5, ax=axes['D'], color=colors)
axes['D'].set_xticks(statistics_df.index, statistics_df.country_year, fontsize=18)
axes['D'].set_ylabel('$\it{SK}$', fontsize=22)
axes['D'].set_xlabel('Umfrageland/-jahr', fontsize=22)
axes['D'].grid(axis='y',color='lightgrey')
axes['D'].spines['top'].set_visible(False)
axes['D'].spines['right'].set_visible(False)

axes['E'].set_title('Standardabweichung des Asset Wealth', fontsize=26, loc='left')
statistics_df.plot(kind='bar', y='mean', width=.5, ax=axes['E'], yerr='std',␣
↪color=colors)
axes['E'].set_xticks(statistics_df.index, statistics_df.country_year, fontsize=18)
axes['E'].set_ylabel('$\it{SD}$', fontsize=22)
axes['E'].set_xlabel('Umfrageland/-jahr', fontsize=22)
axes['E'].grid(axis='y',color='lightgrey')
axes['E'].spines['top'].set_visible(False)
axes['E'].spines['right'].set_visible(False)
axes['E'].get_legend().remove()

axes['F'].set_title('Varianz des Asset Wealth', fontsize=26, loc='left')
statistics_df.plot(kind='bar', y='mean', width=.5, ax=axes['F'], yerr='var',␣
↪color=colors)
axes['F'].get_legend().remove()
axes['F'].set_xticks((statistics_df.index), statistics_df.country_year, fontsize=18)
axes['F'].set_ylabel('$\it{VAR}$', fontsize=22)
axes['F'].set_xlabel('Umfrageland/-jahr', fontsize=22)
axes['F'].grid(axis='y',color='lightgrey')
axes['F'].spines['top'].set_visible(False)
axes['F'].spines['right'].set_visible(False)

fig.legend(handles=[patch2012, patch2015, patch2016], fontsize=18, loc='upper right')
# plt.savefig('./asset_wealth_statistic_analysis.png', dpi=300, bbox_inches='tight', pad_
↪inches = 0)
plt.show()
```




[13]:
```
statistics_df[['country_year', 'mean', 'median', 'std', 'var', 'skewness', 'kurtosis']]
```

[13]:

| | country_year | mean | median | std | var | skewness | kurtosis |
|---|---|---|---|---|---|---|---|
| 0 | COD 2014 | 0.038550 | -0.390554 | 0.958625 | 0.918962 | 2.005833 | 3.031307 |
| 1 | ETH 2016 | 0.012098 | -0.381761 | 0.929815 | 0.864555 | 0.923636 | -0.466208 |
| 2 | KEN 2014 | -0.002535 | -0.040012 | 0.846506 | 0.716573 | 0.031881 | 0.459236 |
| 3 | KEN 2015 | -0.001298 | -0.087786 | 0.775691 | 0.601697 | -0.001202 | 0.677301 |
| 4 | MWI 2014 | 0.003037 | -0.421321 | 0.818947 | 0.670674 | 1.185901 | 0.248446 |
| 5 | MWI 2015 | -0.066928 | -0.355052 | 0.678299 | 0.460089 | 1.820677 | 2.835156 |
| 6 | MWI 2016 | 0.213140 | -0.204208 | 0.868516 | 0.754319 | 1.053283 | 0.232369 |
| 7 | MOZ 2015 | 0.013473 | -0.375884 | 1.469633 | 2.159821 | 0.596114 | -0.635971 |
| 8 | MOZ 2018 | -0.016615 | -0.815378 | 1.832815 | 3.359211 | 0.549163 | -1.223835 |
| 9 | RWA 2014 | -0.151484 | -0.320774 | 0.550205 | 0.302726 | 2.661513 | 7.245296 |
| 10 | RWA 2015 | 0.095847 | -0.292310 | 0.877292 | 0.769642 | 1.595143 | 1.368602 |
| 11 | RWA 2020 | 0.002184 | -0.436307 | 1.324332 | 1.753856 | 1.179800 | 0.752779 |
| 12 | TZA 2015 | 0.145819 | -0.158977 | 0.889375 | 0.790989 | 0.608331 | -0.860105 |
| 13 | TZA 2016 | -0.448258 | -0.642479 | 0.553245 | 0.306080 | 1.637034 | 1.984532 |
| 14 | UGA 2014 | 0.096341 | -0.138693 | 0.836245 | 0.699306 | 1.024133 | 0.407601 |
| 15 | UGA 2015 | -0.147538 | -0.283633 | 0.857924 | 0.736034 | 1.497002 | 1.686292 |
| 16 | UGA 2019 | 0.019539 | -0.389447 | 1.398700 | 1.956362 | 1.135502 | 0.600526 |
| 17 | ZMB 2014 | -0.023153 | -0.377790 | 0.845999 | 0.715714 | 1.141197 | 0.337275 |
| 18 | ZMB 2018 | -0.006915 | -0.685355 | 1.797500 | 3.231007 | 0.795110 | -0.548185 |
| 19 | ZWE 2015 | 0.005259 | -0.367102 | 0.860724 | 0.740845 | 0.196717 | -1.641054 |
| 20 | 2012-2014 | -0.004615 | -0.194061 | 0.848463 | 0.719890 | 0.856188 | 1.222880 |
| 21 | 2015 | 0.022697 | -0.283227 | 0.926333 | 0.858093 | 0.841734 | 0.610966 |
| 22 | 2016-2020 | 0.001080 | -0.415219 | 1.357643 | 1.843195 | 0.966412 | 0.410702 |

### 4.2.2 Predict Asset Wealth for Testset and Mozambique

#### Predict Asset Wealth for Testset

[1]:
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="2"
```

[2]:
```
import sys

sys.path.append("..")

from tensorflow import keras
from keras import optimizers, models

import matplotlib.pyplot as plt
from matplotlib import colors

import pandas as pd
import geopandas as gpd
from pyproj import CRS

import numpy as np
from tqdm.notebook import tqdm
import rasterio

from src.data_utils import combine_wealth_dfs
from src.data_utils import get_label_for_img
from src.data_utils import create_splits

crs = CRS("EPSG:4326")
```

#### Import Data

#### Satellite Images

[3]:
```
viirs_s2_path = '/mnt/datadisk/data/VIIRS_Sentinel2/asset/urban/all/'
viirs_path = '/mnt/datadisk/data/VIIRS/preprocessed/asset/rural//all/'
```

[4]:
```
all_urban_data = os.listdir(viirs_s2_path)
len(all_urban_data)
```

[4]:
```
2458
```

[5]:
```
all_rural_data = os.listdir(viirs_path)
len(all_rural_data)
```

[5]:
```
5602
```

### Geographic Information

[6]:
```python
countries = list(set([x.split('_')[3] for x in all_urban_data]))
countries
```

[6]:
```
['UGA', 'COD', 'MOZ', 'MWI', 'ETH', 'ZMB', 'KEN', 'TZA', 'RWA', 'ZWE']
```

[7]:
```python
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
countries_gdf = world[world.iso_a3.isin(countries)]
countries_gdf
```

[7]:
```
       pop_est continent              name iso_a3  gdp_md_est  \
1     53950935    Africa          Tanzania    TZA    150600.0
11    83301151    Africa  Dem. Rep. Congo    COD     66010.0
13    47615739    Africa             Kenya    KEN    152700.0
48    13805084    Africa          Zimbabwe    ZWE     28330.0
70    15972000    Africa            Zambia    ZMB     65170.0
71    19196246    Africa            Malawi    MWI     21200.0
72    26573706    Africa        Mozambique    MOZ     35010.0
165  105350020    Africa          Ethiopia    ETH    174700.0
168   39570125    Africa            Uganda    UGA     84930.0
169   11901484    Africa            Rwanda    RWA     21970.0

                                              geometry
1    POLYGON ((33.90371 -0.95000, 34.07262 -1.05982...
11   POLYGON ((29.34000 -4.49998, 29.51999 -5.41998...
13   POLYGON ((39.20222 -4.67677, 37.76690 -3.67712...
48   POLYGON ((31.19141 -22.25151, 30.65987 -22.151...
70   POLYGON ((30.74001 -8.34001, 31.15775 -8.59458...
71   POLYGON ((32.75938 -9.23060, 33.73972 -9.41715...
72   POLYGON ((34.55999 -11.52002, 35.31240 -11.439...
165  POLYGON ((47.78942 8.00300, 44.96360 5.00162, ...
168  POLYGON ((33.90371 -0.95000, 31.86617 -1.02736...
169  POLYGON ((30.41910 -1.13466, 30.81613 -1.69891...
```

### Urban Data

[8]:
```python
X_train_val, X_test, y_train_val, y_test = create_splits(img_dir='/mnt/datadisk/data/
↪VIIRS_Sentinel2/asset/urban/all/', wealth_path='/home/stoermer/Sentinel/gps_csv/',␣
↪urban_rural='u', pre2015_path=False)
```

```
Gathering Label Data
```

```
100%|| 2458/2458 [01:03<00:00, 38.76it/s]
```

[9]:
```python
wealth_df = combine_wealth_dfs('/home/stoermer/Sentinel/gps_csv/')

urban_test_labels = pd.DataFrame()
for x in tqdm(X_test):
    urban_test_labels = pd.concat([urban_test_labels, get_label_for_img(wealth_df, x)])
urban_test_labels
```

```
  0%|          | 0/492 [00:00<?, ?it/s]
```

[9]:

```
   WEALTH_INDEX  SURVEY_YEAR   LATNUM  LONGNUM
0      1.071443         2015 -17.9025  30.6523
0     -0.068987         2014  -9.8081  29.0395
0      2.655006         2013  -4.3835  15.3125
0      1.668783         2019   2.7515  32.2981
0      1.278176         2015 -20.1791  28.6178
..          ...          ...      ...      ...
0      0.780936         2016  13.3427  39.7597
0      0.980313         2016  -5.7441  34.8325
0      0.135483         2019  -2.7007  29.0006
0      2.779817         2018 -25.8828  32.5081
0      0.327767         2016   7.2420  37.8950

[492 rows x 4 columns]
```

### Get Predictions of Best Model

[10]:

```
model = keras.models.load_model('./vgg19_viirs_s2_u.h5')
```

[11]:

```
data_urban = np.zeros(shape=(len(X_test), 14, 200, 200))
for index, img in tqdm(enumerate(X_test)):
    # Read in each Image
    with rasterio.open(os.path.join(viirs_s2_path, img)) as i:
        array = i.read().astype("float32")

    # Ensure that the Array is not empty
    array[np.isnan(array)] = 0
    assert not np.any(np.isnan(array)), "Float"

    # Add to batch
    data_urban[index] = array

    # Check if batch is already full (Note: Index in batch array is from 0...4 hence we␣
↪need to add +1 to batch_ele)
data_urban = data_urban.transpose(0, 2, 3, 1)
preds_urban = model.predict(data_urban)
len(preds_urban)
```

```
0it [00:00, ?it/s]
```

[11]:

```
492
```

### Create Geo DataFrames for Groundtruth and Predictions

[12]:
```
geometry = gpd.points_from_xy(urban_test_labels.LONGNUM, urban_test_labels.LATNUM)

true_urban_wealth_df = gpd.GeoDataFrame(urban_test_labels,
                        geometry=geometry,
                        crs=crs
                       )
true_urban_wealth_df['COUNTRY_CODE'] = [x.split('_')[3] for x in X_test]
true_urban_wealth_df = true_urban_wealth_df[['SURVEY_YEAR', 'geometry','COUNTRY_CODE',
↪'WEALTH_INDEX']]
true_urban_wealth_df.head(3)
```

[12]:
```
   SURVEY_YEAR                     geometry COUNTRY_CODE  WEALTH_INDEX
0         2015  POINT (30.65230 -17.90250)          ZWE      1.071443
0         2014   POINT (29.03950 -9.80810)          ZMB     -0.068987
0         2013   POINT (15.31250 -4.38350)          COD      2.655006
```

[13]:
```
predicted_urban_wealth_df = true_urban_wealth_df.loc[:,['SURVEY_YEAR', 'geometry']]
predicted_urban_wealth_df['WEALTH_INDEX'] = preds_urban
predicted_urban_wealth_df
```

[13]:
```
    SURVEY_YEAR                     geometry  WEALTH_INDEX
0          2015  POINT (30.65230 -17.90250)      1.538016
0          2014   POINT (29.03950 -9.80810)      0.407157
0          2013   POINT (15.31250 -4.38350)      2.204992
0          2019    POINT (32.29810 2.75150)      1.057997
0          2015  POINT (28.61780 -20.17910)      0.996908
..          ...                          ...           ...
0          2016   POINT (39.75970 13.34270)      0.425004
0          2016   POINT (34.83250 -5.74410)      1.155177
0          2019   POINT (29.00060 -2.70070)      0.764184
0          2018  POINT (32.50810 -25.88280)      1.686300
0          2016    POINT (37.89500 7.24200)      0.759801

[492 rows x 3 columns]
```

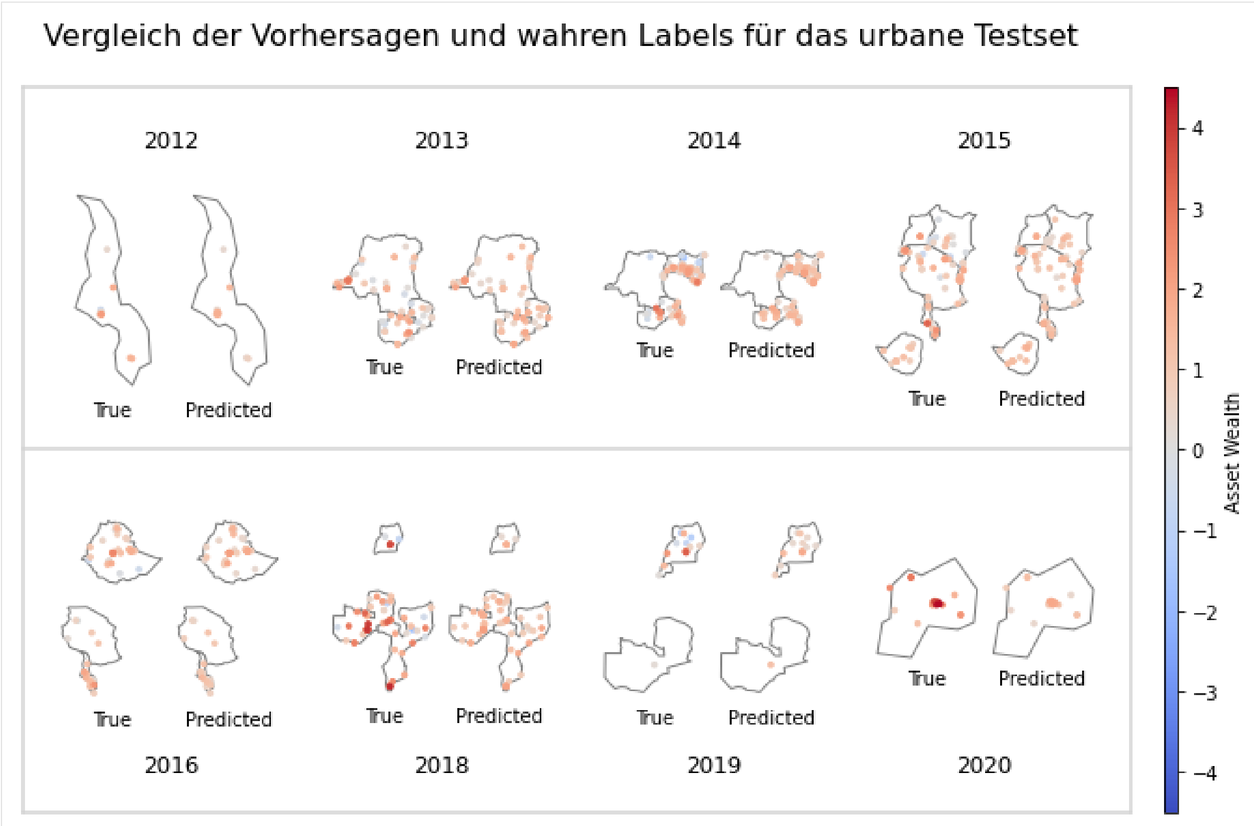
[14]:
```
crs = CRS("EPSG:4326")
predicted_urban_wealth_df = gpd.GeoDataFrame(predicted_urban_wealth_df,
                                             geometry=predicted_urban_wealth_df['geometry
↪'],
                                             crs=crs)
true_urban_wealth_df = gpd.GeoDataFrame(true_urban_wealth_df,
                                        geometry=true_urban_wealth_df['geometry'],
                                        crs=crs)
```

### Plot Predictions vs. Groundtruth

[15]:
```
divnorm=colors.TwoSlopeNorm(vmin=-4.5, vcenter=0., vmax=4.5)
# plt.subplots_adjust(wspace=None, hspace=None)
markersize=7
predicted_countries_gdf = countries_gdf[countries_gdf.iso_a3.isin(true_urban_wealth_df.
↪COUNTRY_CODE)]

mosaic =  [['title_2012','title_2012','title_2012','title_2012','title_2012','title_2012
↪','.',
            'title_2013','title_2013','title_2013','title_2013','title_2013','title_2013
↪','.',
            'title_2014','title_2014','title_2014','title_2014','title_2014','title_2014
↪','.',
            'title_2015','title_2015','title_2015','title_2015','title_2015','title_2015
↪'],
        ['true2012','true2012','true2012','pred2012','pred2012','pred2012','.',
            'true2013', 'true2013','true2013','pred2013','pred2013','pred2013', '.',
            'true2014','true2014','true2014','pred2014','pred2014','pred2014','.',
        'true2015','true2015','true2015','pred2015','pred2015','pred2015'],
           ['title_2016','title_2016','title_2016','title_2016','title_2016','title_2016
↪','.',
            'title_2018','title_2018','title_2018','title_2018','title_2018','title_2018
↪','.',
            'title_2019','title_2019','title_2019','title_2019','title_2019','title_2019
↪','.',
            'title_2020','title_2020','title_2020','title_2020','title_2020','title_2020
↪'],
           ['true2016','true2016','true2016','pred2016','pred2016','pred2016', '.',
            'true2018','true2018','true2018', 'pred2018', 'pred2018','pred2018','.',
            'true2019','true2019','true2019','pred2019','pred2019','pred2019','.',
            'true2020', 'true2020', 'true2020','pred2020','pred2020','pred2020'],
]

fig, axes = plt.subplot_mosaic(mosaic,figsize=(10,6), gridspec_kw=(dict(height_ratios=(.
↪8,3,.8,3))))#, constrained_layout=True)
fig.suptitle('Vergleich der Vorhersagen und wahren Labels für das urbane Testset',
↪fontsize=16)

for index, row in enumerate(mosaic):
    row = [k for k in row if k!='.']
    keys=[]
    for k in row:
        if k not in keys:
            keys.append(k)
#     print(keys)
    for k in keys:
        predicted_countries_gdf = countries_gdf[countries_gdf.iso_a3.isin(true_urban_
↪wealth_df[true_urban_wealth_df.SURVEY_YEAR==int(k[-4:])].COUNTRY_CODE)]
        if k.startswith('title'):
            if index==0:
                axes[k].set_title(k[-4:], fontsize=12,y=0)
```
(continues on next page)

(continued from previous page)

```
            else:
                axes[k].set_title(k[-4:], fontsize=12,y=-5.5)
        elif k.startswith('true'):
            predicted_countries_gdf.plot(ax=axes[k], color='white', edgecolor='grey')
            true_urban_wealth_df[true_urban_wealth_df.SURVEY_YEAR==int(k[-4:])].sort_
↪values(by='WEALTH_INDEX').plot(ax = axes[k], column='WEALTH_INDEX', cmap='coolwarm',
↪norm=divnorm, markersize=markersize)
            axes[k].set_xlabel('True')
        else:
            predicted_countries_gdf.plot(ax=axes[k], color='white', edgecolor='grey')
            predicted_urban_wealth_df[predicted_urban_wealth_df.SURVEY_YEAR==int(k[-4:
↪])].sort_values(by='WEALTH_INDEX').plot(ax = axes[k], column='WEALTH_INDEX', cmap=
↪'coolwarm',norm=divnorm, markersize=markersize)
            axes[k].set_xlabel('Predicted')
        axes[k].spines['top'].set_visible(False)
        axes[k].spines['right'].set_visible(False)
        axes[k].spines['bottom'].set_visible(False)
        axes[k].spines['left'].set_visible(False)
        axes[k].get_xaxis().set_ticks([])
        axes[k].get_yaxis().set_ticks([])

line = plt.Line2D([0.1,.925],[.45,.45], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,.925],[0,0], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,.925],[.9,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,0.1],[0,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([.925,.925],[0,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)

sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=-4.5, vmax=4.5), cmap='coolwarm')
cbaxes = fig.add_axes([0.95, 0,  0.01, 0.9])
cbar = fig.colorbar(sm, orientation='vertical',label='Asset Wealth',cax=cbaxes)

plt.show()
```




### Rural Data

### Get Label Data

[16]:
```
wealth_df = combine_wealth_dfs('/home/stoermer/Sentinel/gps_csv/')
```

[17]:
```
X_train_val, X_test, y_train_val, y_test = create_splits(img_dir='/mnt/datadisk/data/
↪VIIRS/preprocessed/asset/rural/all/', wealth_path='/home/stoermer/Sentinel/gps_csv/',␣
↪urban_rural='r', pre2015_path=False)
```

```
Gathering Label Data
```

```
100%|| 5602/5602 [02:25<00:00, 38.55it/s]
```

[18]:
```
rural_test_labels = pd.DataFrame()
for x in tqdm(X_test):
    rural_test_labels = pd.concat([rural_test_labels, get_label_for_img(wealth_df, x)])
rural_test_labels
```

```
  0%|          | 0/1121 [00:00<?, ?it/s]
```

[18]:
```
   WEALTH_INDEX  SURVEY_YEAR    LATNUM  LONGNUM
0     -0.688140         2015  -17.7214  31.7259
0     -0.216390         2014   -0.6815  35.2460
0     -0.112214         2016    7.2301  35.3157
```

(continues on next page)

(continued from previous page)

```
0      -0.212770         2015  -12.9103  34.2767
0      -0.168055         2016   -8.4875  39.2632
..           ...          ...       ...      ...
0      -0.734604         2014    1.9172  33.6136
0      -0.500560         2015   -1.7555  29.6134
0      -0.504263         2015  -16.4775  30.4724
0      -0.375565         2014    0.0005  37.9925
0      -0.537768         2015   -4.4555  39.2872

[1121 rows x 4 columns]
```

### Get Predictions of Best Model

[19]:
```
model = keras.models.load_model('./resnet50_r_viirs.h5')
```

[20]:
```
data_rural = np.zeros(shape=(len(X_test), 3, 1000, 1000))
for index, img in tqdm(enumerate(X_test)):
    # Read in each Image
    with rasterio.open(os.path.join(viirs_path, img)) as i:
        array = i.read().astype("float32")

    # Ensure that the Array is not empty
    array[np.isnan(array)] = 0
    assert not np.any(np.isnan(array)), "Float"

    # Add to batch
    data_rural[index] = array

    # Check if batch is already full (Note: Index in batch array is from 0...4 hence we␣
↪need to add +1 to batch_ele)
data_rural = data_rural.transpose(0, 2, 3, 1)
preds_rural = model.predict(data_rural)
len(preds_rural)
```

```
0it [00:00, ?it/s]
```

[20]:
```
1121
```

### Create Geo DataFrames for Groundtruth and Predictions

[21]:
```
crs = CRS("EPSG:4326")
geometry = gpd.points_from_xy(rural_test_labels.LONGNUM, rural_test_labels.LATNUM)

true_rural_wealth_df = gpd.GeoDataFrame(rural_test_labels,
                        geometry=geometry,
                        crs=crs
                       )
true_rural_wealth_df['COUNTRY_CODE'] = [x.split('_')[3] for x in X_test]
```

(continues on next page)

(continued from previous page)

```
true_rural_wealth_df = true_rural_wealth_df[['SURVEY_YEAR', 'geometry', 'COUNTRY_CODE',
↪'WEALTH_INDEX']]
true_rural_wealth_df.head(3)
```

[21]:

```
   SURVEY_YEAR                     geometry COUNTRY_CODE  WEALTH_INDEX
0         2015  POINT (31.72590 -17.72140)          ZWE     -0.688140
0         2014   POINT (35.24600 -0.68150)          KEN     -0.216390
0         2016    POINT (35.31570 7.23010)          ETH     -0.112214
```

[22]:

```
predicted_rural_wealth_df = true_rural_wealth_df.loc[:,['SURVEY_YEAR', 'geometry',
↪'COUNTRY_CODE']]
predicted_rural_wealth_df['WEALTH_INDEX'] = preds_rural
predicted_rural_wealth_df.head(3)
```

[22]:

```
   SURVEY_YEAR                     geometry COUNTRY_CODE  WEALTH_INDEX
0         2015  POINT (31.72590 -17.72140)          ZWE     -0.548155
0         2014   POINT (35.24600 -0.68150)          KEN     -0.588530
0         2016    POINT (35.31570 7.23010)          ETH     -0.588530
```

### Plot Predictions vs. Groundtruth

[23]:

```
divnorm=colors.TwoSlopeNorm(vmin=-4.5, vcenter=0., vmax=4.5)
# plt.subplots_adjust(wspace=None, hspace=None)
markersize=7
predicted_countries_gdf = countries_gdf[countries_gdf.iso_a3.isin(true_rural_wealth_df.
↪COUNTRY_CODE)]

mosaic =  [['title_2012','title_2012','title_2012','title_2012','title_2012','title_2012
↪','.',
            'title_2013','title_2013','title_2013','title_2013','title_2013','title_2013
↪','.',
            'title_2014','title_2014','title_2014','title_2014','title_2014','title_2014
↪','.',
            'title_2015','title_2015','title_2015','title_2015','title_2015','title_2015
↪'],
        ['true2012','true2012','true2012','pred2012','pred2012','pred2012','.',
            'true2013', 'true2013','true2013','pred2013','pred2013','pred2013', '.',
            'true2014','true2014','true2014','pred2014','pred2014','pred2014','.',
        'true2015','true2015','true2015','pred2015','pred2015','pred2015'],
           ['title_2016','title_2016','title_2016','title_2016','title_2016','title_2016
↪','.',
            'title_2018','title_2018','title_2018','title_2018','title_2018','title_2018
↪','.',
            'title_2019','title_2019','title_2019','title_2019','title_2019','title_2019
↪','.',
            'title_2020','title_2020','title_2020','title_2020','title_2020','title_2020
↪'],
           ['true2016','true2016','true2016','pred2016','pred2016','pred2016', '.',
            'true2018','true2018','true2018', 'pred2018', 'pred2018','pred2018','.',
            'true2019','true2019','true2019','pred2019','pred2019','pred2019','.',
            'true2020', 'true2020', 'true2020','pred2020','pred2020','pred2020'],
```

(continues on next page)

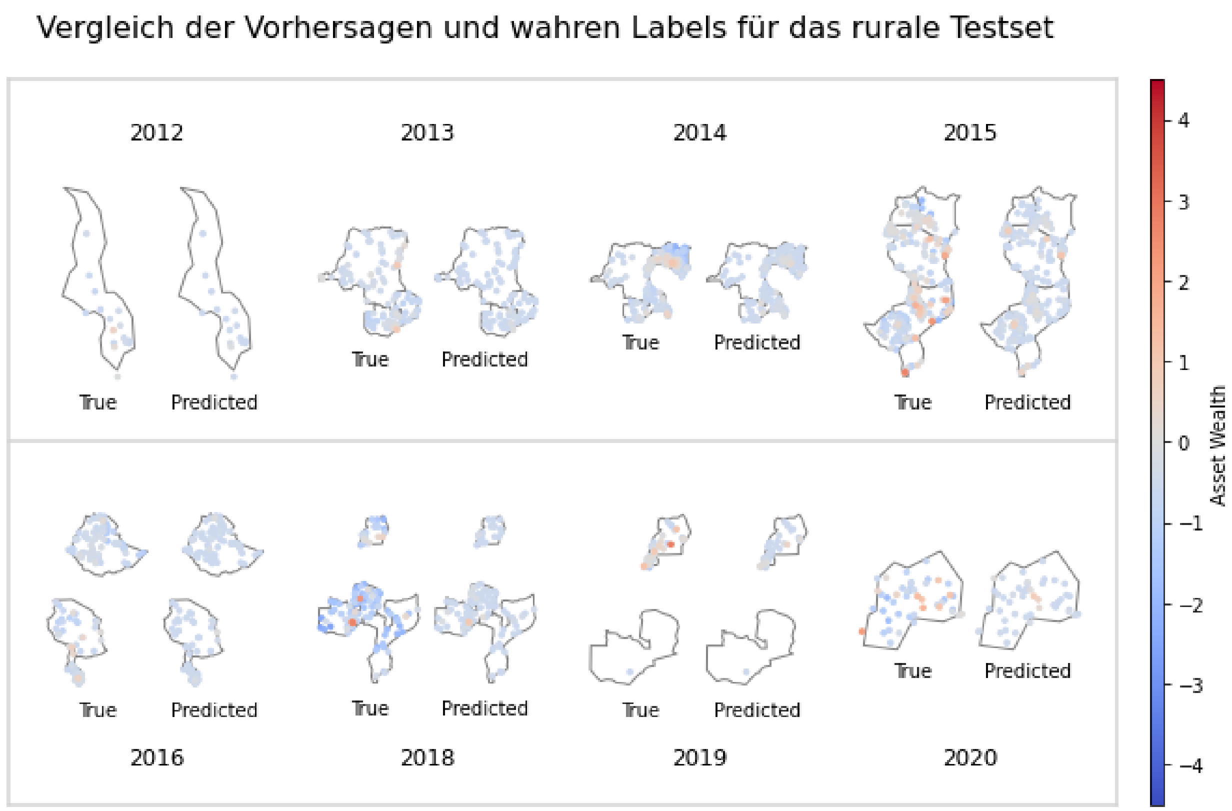
(continued from previous page)

```
]

fig, axes = plt.subplot_mosaic(mosaic,figsize=(10,6), gridspec_kw=(dict(height_ratios=(.
↪8,3,.8,3))))#, constrained_layout=True)
fig.suptitle('Vergleich der Vorhersagen und wahren Labels für das rurale Testset',
↪fontsize=16)

for index, row in enumerate(mosaic):
    row = [k for k in row if k!='.']
    keys=[]
    for k in row:
        if k not in keys:
            keys.append(k)
#     print(keys)
    for k in keys:
        predicted_countries_gdf = countries_gdf[countries_gdf.iso_a3.isin(true_rural_
↪wealth_df[true_rural_wealth_df.SURVEY_YEAR==int(k[-4:])].COUNTRY_CODE)]
        if k.startswith('title'):
            if index==0:
                axes[k].set_title(k[-4:], fontsize=12,y=0)
            else:
                axes[k].set_title(k[-4:], fontsize=12,y=-5.5)
        elif k.startswith('true'):
            predicted_countries_gdf.plot(ax=axes[k], color='white', edgecolor='grey')
            true_rural_wealth_df[true_rural_wealth_df.SURVEY_YEAR==int(k[-4:])].sort_
↪values(by='WEALTH_INDEX').plot(ax = axes[k], column='WEALTH_INDEX', cmap='coolwarm',
↪norm=divnorm, markersize=markersize)
            axes[k].set_xlabel('True')
        else:
            predicted_countries_gdf.plot(ax=axes[k], color='white', edgecolor='grey')
            predicted_rural_wealth_df[predicted_rural_wealth_df.SURVEY_YEAR==int(k[-4:
↪])].sort_values(by='WEALTH_INDEX').plot(ax = axes[k], column='WEALTH_INDEX', cmap=
↪'coolwarm',norm=divnorm, markersize=markersize)
            axes[k].set_xlabel('Predicted')
        axes[k].spines['top'].set_visible(False)
        axes[k].spines['right'].set_visible(False)
        axes[k].spines['bottom'].set_visible(False)
        axes[k].spines['left'].set_visible(False)
        axes[k].get_xaxis().set_ticks([])
        axes[k].get_yaxis().set_ticks([])

line = plt.Line2D([0.1,.925],[.45,.45], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,.925],[0,0], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,.925],[.9,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,0.1],[0,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([.925,.925],[0,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
```

(continues on next page)

(continued from previous page)

```
sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=-4.5, vmax=4.5), cmap='coolwarm')
cbaxes = fig.add_axes([0.95, 0,  0.01, 0.9])
cbar = fig.colorbar(sm, orientation='vertical',label='Asset Wealth',cax=cbaxes)

plt.show()
```




### Combine rural and urban predictions and labels

```
[24]: predicted_wealth_df = pd.concat([predicted_rural_wealth_df,predicted_urban_wealth_df])
true_wealth_df = pd.concat([true_rural_wealth_df, true_urban_wealth_df])
```

### Plot all predictions

```
[25]: divnorm=colors.TwoSlopeNorm(vmin=-4.5, vcenter=0., vmax=4.5)
markersize=5
predicted_countries_gdf = countries_gdf[countries_gdf.iso_a3.isin(true_rural_wealth_df.
↪COUNTRY_CODE)]

mosaic =  [['title_2012','title_2012','title_2012','title_2012','title_2012','title_2012
↪','.',
           'title_2013','title_2013','title_2013','title_2013','title_2013','title_2013
↪','.',
           'title_2014','title_2014','title_2014','title_2014','title_2014','title_2014
↪','.',
```

(continues on next page)

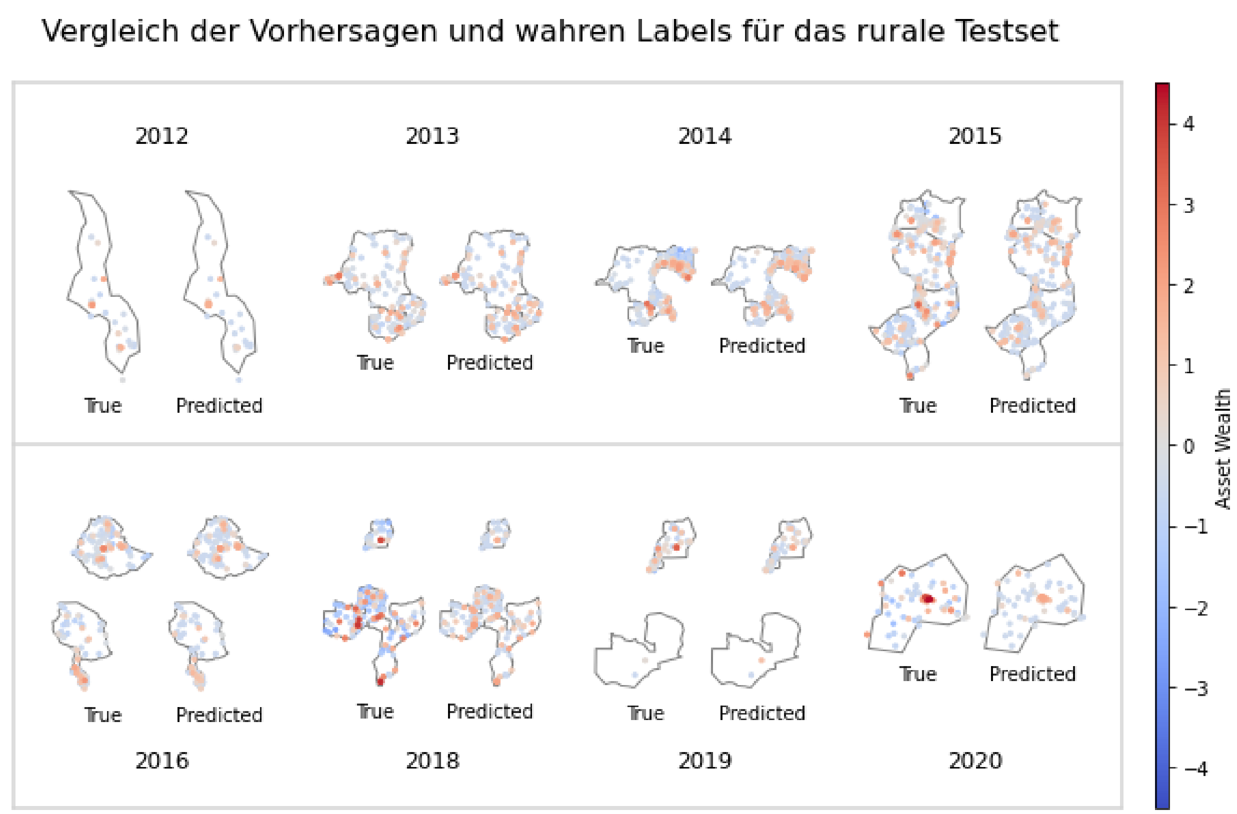
(continued from previous page)

```
             'title_2015','title_2015','title_2015','title_2015','title_2015','title_2015
↪'],
         ['true2012','true2012','true2012','pred2012','pred2012','pred2012','.',
             'true2013', 'true2013','true2013','pred2013','pred2013','pred2013', '.',
             'true2014','true2014','true2014','pred2014','pred2014','pred2014','.',
         'true2015','true2015','true2015','pred2015','pred2015','pred2015'],
            ['title_2016','title_2016','title_2016','title_2016','title_2016','title_2016
↪','.',
             'title_2018','title_2018','title_2018','title_2018','title_2018','title_2018
↪','.',
             'title_2019','title_2019','title_2019','title_2019','title_2019','title_2019
↪','.',
             'title_2020','title_2020','title_2020','title_2020','title_2020','title_2020
↪'],
            ['true2016','true2016','true2016','pred2016','pred2016','pred2016', '.',
             'true2018','true2018','true2018', 'pred2018', 'pred2018','pred2018','.',
             'true2019','true2019','true2019','pred2019','pred2019','pred2019','.',
             'true2020', 'true2020', 'true2020','pred2020','pred2020','pred2020'],
]

fig, axes = plt.subplot_mosaic(mosaic,figsize=(10,6), gridspec_kw=(dict(height_ratios=(.
↪8,3,.8,3))))#, constrained_layout=True)
fig.suptitle('Vergleich der Vorhersagen und wahren Labels für das rurale Testset',
↪fontsize=16)

for index, row in enumerate(mosaic):
    row = [k for k in row if k!='.']
    keys=[]
    for k in row:
        if k not in keys:
            keys.append(k)

    for k in keys:
        predicted_countries_gdf = countries_gdf[countries_gdf.iso_a3.isin(true_wealth_
↪df[true_wealth_df.SURVEY_YEAR==int(k[-4:])].COUNTRY_CODE)]
        if k.startswith('title'):
            if index==0:
                axes[k].set_title(k[-4:], fontsize=12,y=0)
            else:
                axes[k].set_title(k[-4:], fontsize=12,y=-5.5)
        elif k.startswith('true'):
            predicted_countries_gdf.plot(ax=axes[k], color='white', edgecolor='grey')
            true_wealth_df[true_wealth_df.SURVEY_YEAR==int(k[-4:])].sort_values(by=
↪'WEALTH_INDEX').plot(ax = axes[k], column='WEALTH_INDEX', cmap='coolwarm',norm=divnorm,
↪ markersize=markersize)
            axes[k].set_xlabel('True')
        else:
            predicted_countries_gdf.plot(ax=axes[k], color='white', edgecolor='grey')
            predicted_wealth_df[predicted_wealth_df.SURVEY_YEAR==int(k[-4:])].sort_
↪values(by='WEALTH_INDEX').plot(ax = axes[k], column='WEALTH_INDEX', cmap='coolwarm',
↪norm=divnorm, markersize=markersize)
            axes[k].set_xlabel('Predicted')
```

(continues on next page)

(continued from previous page)

```
        axes[k].spines['top'].set_visible(False)
        axes[k].spines['right'].set_visible(False)
        axes[k].spines['bottom'].set_visible(False)
        axes[k].spines['left'].set_visible(False)
        axes[k].get_xaxis().set_ticks([])
        axes[k].get_yaxis().set_ticks([])

line = plt.Line2D([0.1,.925],[.45,.45], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,.925],[0,0], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,.925],[.9,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([0.1,0.1],[0,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)
line = plt.Line2D([.925,.925],[0,.9], transform=fig.transFigure, color="lightgrey")
fig.add_artist(line)

sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=-4.5, vmax=4.5), cmap='coolwarm')
cbaxes = fig.add_axes([0.95, 0,  0.01, 0.9])
cbar = fig.colorbar(sm, orientation='vertical',label='Asset Wealth',cax=cbaxes)

plt.show()
```



### Get r2 Score for best Models

[13]:
```
from sklearn.metrics import mean_squared_error, r2_score


urban_r2_score = r2_score(true_urban_wealth_df.WEALTH_INDEX, predicted_urban_wealth_df.
↪WEALTH_INDEX)
rural_r2_score = r2_score(true_rural_wealth_df.WEALTH_INDEX, predicted_rural_wealth_df.
↪WEALTH_INDEX)

print(urban_r2_score)
rural_r2_score
```

```
0.32608796277450536
```

[13]:
```
0.14404635936378374
```

### Save Predictions and Groundtruth

[26]:
```
predicted_urban_wealth_df.to_csv('./predicted_urban_wealth_df.csv')
true_urban_wealth_df.to_csv('./true_urban_wealth_df.csv')

predicted_rural_wealth_df.to_csv('./predicted_rural_wealth_df.csv')
true_rural_wealth_df.to_csv('./true_rural_wealth_df.csv')
```

[6]:
```
predicted_urban_wealth_df = pd.read_csv('./predicted_urban_wealth_df.csv')[['SURVEY_YEAR
↪','geometry', 'WEALTH_INDEX']]
predicted_urban_wealth_df = gpd.GeoDataFrame(predicted_urban_wealth_df.loc[:, [c for c␣
↪in predicted_urban_wealth_df.columns if c != "geometry"]],
                                            geometry=gpd.GeoSeries.from_wkt(predicted_
↪urban_wealth_df["geometry"]),
                                            crs=crs,
                                           )

true_urban_wealth_df = pd.read_csv('./true_urban_wealth_df.csv')[['SURVEY_YEAR','geometry
↪','COUNTRY_CODE', 'WEALTH_INDEX']]
true_urban_wealth_df = gpd.GeoDataFrame(true_urban_wealth_df.loc[:, [c for c in true_
↪urban_wealth_df.columns if c != "geometry"]],
                                       geometry=gpd.GeoSeries.from_wkt(true_urban_
↪wealth_df["geometry"]),
                                       crs=crs,
                                      )



predicted_rural_wealth_df = pd.read_csv('./predicted_rural_wealth_df.csv')[['SURVEY_YEAR
↪','geometry', 'WEALTH_INDEX']]
predicted_rural_wealth_df = gpd.GeoDataFrame(predicted_rural_wealth_df.loc[:, [c for c␣
↪in predicted_rural_wealth_df.columns if c != "geometry"]],
                                            geometry=gpd.GeoSeries.from_wkt(predicted_
↪rural_wealth_df["geometry"]),
                                            crs=crs,
                                           )
```

(continues on next page)

(continued from previous page)

```
true_rural_wealth_df = pd.read_csv('./true_rural_wealth_df.csv')[['SURVEY_YEAR','geometry
↪','COUNTRY_CODE', 'WEALTH_INDEX']]
true_rural_wealth_df = gpd.GeoDataFrame(true_rural_wealth_df.loc[:, [c for c in true_
↪rural_wealth_df.columns if c != "geometry"]],
                                        geometry=gpd.GeoSeries.from_wkt(true_rural_
↪wealth_df["geometry"]),
                                        crs=crs,
                                       )
```

### Predict Asset Wealth for Mozambique

### Load general geographic Data of Mozambique

[28]:
```
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
moz = world[world.name=='Mozambique']
```

### Load Mozambique Survey Data

[29]:
```
moz_df = pd.read_csv('./moz_2016_2021.csv')
moz_df = moz_df[['URBAN_RURA', 'LATNUM', 'LONGNUM', 'SURVEY_YEAR', 'Filename']]
print(moz_df.shape)
moz_df.head(3)
```

```
(2635, 5)
```

[29]:
```
   URBAN_RURA    LATNUM  LONGNUM  SURVEY_YEAR  \
0           0 -13.2856  35.2311         2016
1           0 -13.3226  35.2552         2016
2           0 -13.2716  35.1965         2016

                                        Filename
0  -13.2856_35.2311_20160101-20161231_MOZ_u_2.0.tif
1  -13.3226_35.2552_20160101-20161231_MOZ_u_2.0.tif
2  -13.2716_35.1965_20160101-20161231_MOZ_u_2.0.tif
```

### Urban

### Load Data for Mozambique 2016, 2017, 2019, 2020 and 2021

[30]:
```
model = keras.models.load_model('./vgg19_viirs_s2_u.h5')
```

[31]:
```
moz_path_urban = '/mnt/datadisk/data/VIIRS_Sentinel2/asset/urban/mozambique_2016_2021'
moz_list_urban = os.listdir(moz_path_urban)
len(moz_list_urban)
```

[31]:
```
1115
```

[32]:
```
moz_2016_urban = [i for i in moz_list_urban if '2016' in i]
moz_2017_urban = [i for i in moz_list_urban if '2017' in i]
moz_2019_urban = [i for i in moz_list_urban if '2019' in i]
moz_2020_urban = [i for i in moz_list_urban if '2020' in i]
moz_2021_urban = [i for i in moz_list_urban if '2021' in i]
```

[33]:
```
data_urban = {}
for moz_data_urban in [[moz_2016_urban,'2016'],[moz_2017_urban,'2017'],[moz_2019_urban,
↪'2019'],[moz_2020_urban,'2020'],[moz_2021_urban,'2021']]:
    data_urban[moz_data_urban[1]] = np.zeros(shape=(len(moz_data_urban[0]), 14, 200,␣
↪200))
    for index, img in tqdm(enumerate(moz_data_urban[0])):
        # Read in each Image
        with rasterio.open(os.path.join(moz_path_urban, img)) as i:
            array = i.read().astype("float32")

        # Ensure that the Array is not empty
        array[np.isnan(array)] = 0
        assert not np.any(np.isnan(array)), "Float"

        data_urban[moz_data_urban[1]][index] = array

    data_urban[moz_data_urban[1]] = data_urban[moz_data_urban[1]].transpose(0, 2, 3, 1)
    assert len(data_urban[moz_data_urban[1]]) == len(moz_data_urban[0])
```

```
0it [00:00, ?it/s]
```

```
0it [00:00, ?it/s]
```

```
0it [00:00, ?it/s]
```

```
0it [00:00, ?it/s]
```

```
0it [00:00, ?it/s]
```

### Predict Asset Wealth

[34]:
```
preds_urban = {}
for year in tqdm(data_urban.keys()):
    preds_urban[year] = model.predict(data_urban[year])
print(preds_urban.keys())
len(preds_urban['2016'])
```

```
  0%|          | 0/5 [00:00<?, ?it/s]
```

```
dict_keys(['2016', '2017', '2019', '2020', '2021'])
```

[34]:
```
223
```

[35]:
```
pred_file_urban_dict = {'Filename':moz_2016_urban+moz_2017_urban+moz_2019_urban+moz_2020_
↪urban+moz_2021_urban,
                    'Prediction':[pred[0] for pred in preds_urban['2016']]+
                    [pred[0] for pred in preds_urban['2017']]+
                    [pred[0] for pred in preds_urban['2019']]+
```

(continues on next page)

(continued from previous page)

```
                  [pred[0] for pred in preds_urban['2020']]+
                  [pred[0] for pred in preds_urban['2021']]}
pred_file_urban_df = pd.DataFrame(pred_file_urban_dict)
print(pred_file_urban_df.shape)
pred_file_urban_df.head(3)
```

```
(1115, 2)
```

[35]:
```
                                             Filename  Prediction
0  -25.9443_32.6146_20160101-20161231_MOZ_u_2.0.tif    1.961596
1  -25.9002_32.6141_20160101-20161231_MOZ_u_2.0.tif    1.722382
2  -23.7538_35.3463_20160101-20161231_MOZ_u_2.0.tif    1.993611
```

### Merge Predictions with Survey Data

[36]:
```
asset_pred_urban_df = moz_df.merge(pred_file_urban_df)
print(asset_pred_urban_df.shape)
asset_pred_urban_df.head(3)
```

```
(1115, 6)
```

[36]:
```
   URBAN_RURA    LATNUM  LONGNUM  SURVEY_YEAR  \
0           0 -13.2856  35.2311         2016
1           0 -13.3226  35.2552         2016
2           0 -13.2716  35.1965         2016

                                             Filename  Prediction
0  -13.2856_35.2311_20160101-20161231_MOZ_u_2.0.tif    1.521431
1  -13.3226_35.2552_20160101-20161231_MOZ_u_2.0.tif    1.153307
2  -13.2716_35.1965_20160101-20161231_MOZ_u_2.0.tif    1.414657
```

### Merge Predictions with Survey Data

[37]:
```
geometry = gpd.points_from_xy(asset_pred_urban_df.LONGNUM, asset_pred_urban_df.LATNUM)
```

[38]:
```
asset_pred_urban_df = gpd.GeoDataFrame(asset_pred_urban_df,
                       geometry=geometry,
                       crs=crs
                      )
asset_pred_urban_df = asset_pred_urban_df[['SURVEY_YEAR', 'geometry', 'Prediction']]
print(asset_pred_urban_df.shape)
asset_pred_urban_df.head(3)
```

```
(1115, 3)
```

[38]:
```
   SURVEY_YEAR                    geometry  Prediction
0         2016  POINT (35.23110 -13.28560)    1.521431
1         2016  POINT (35.25520 -13.32260)    1.153307
2         2016  POINT (35.19650 -13.27160)    1.414657
```

[39]:
```
urban_2016_df = asset_pred_urban_df[asset_pred_urban_df.SURVEY_YEAR==2016]
urban_2016_df.shape

urban_2017_df = asset_pred_urban_df[asset_pred_urban_df.SURVEY_YEAR==2017]
urban_2017_df.shape

urban_2019_df = asset_pred_urban_df[asset_pred_urban_df.SURVEY_YEAR==2019]
urban_2019_df.shape

urban_2020_df = asset_pred_urban_df[asset_pred_urban_df.SURVEY_YEAR==2020]
urban_2020_df.shape

urban_2021_df = asset_pred_urban_df[asset_pred_urban_df.SURVEY_YEAR==2021]
urban_2021_df.shape
```

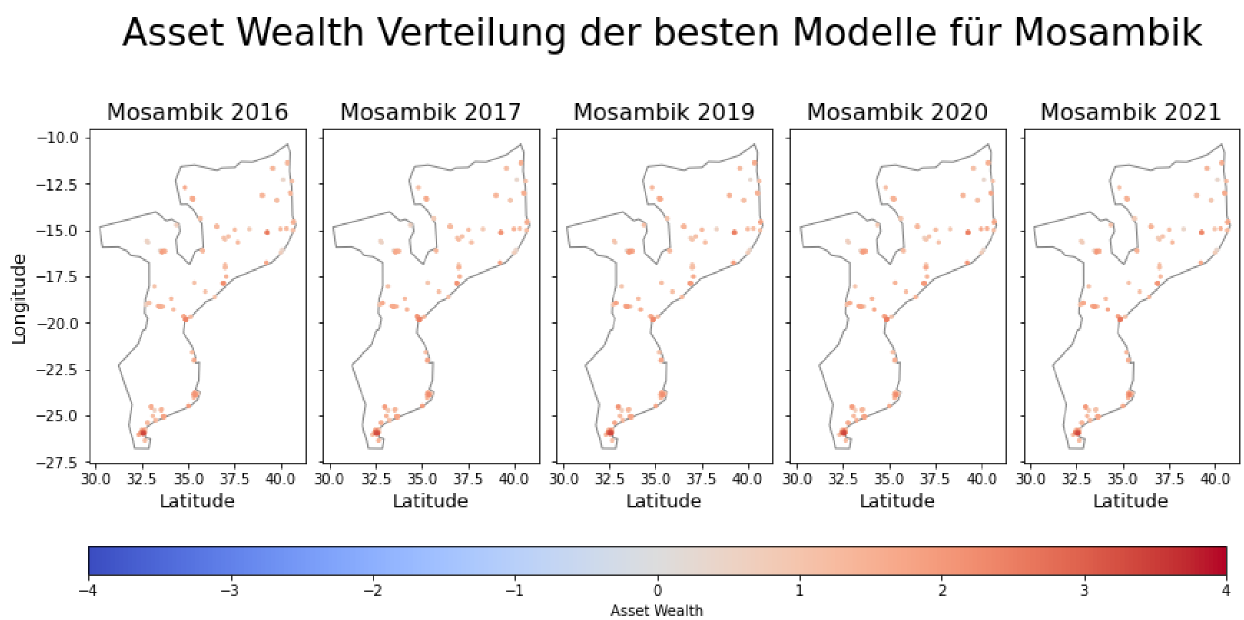
[39]:
```
(223, 3)
```

[40]:
```
fig, axes = plt.subplots(1, 5, sharey=True,constrained_layout=True, figsize=(12,9))


from matplotlib import colors

divnorm=colors.TwoSlopeNorm(vmin=-4, vcenter=0, vmax=4)
markersize=5
subplot_title_size = 16

axes[0].set_aspect('equal')
plt.suptitle('Asset Wealth Verteilung der besten Modelle für Mosambik', fontsize=26, y=.
↪8, x=0.53)

moz.plot(ax=axes[0], color='white', edgecolor='grey')

urban_2016_df.sort_values(by='Prediction').plot(ax = axes[0], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)


axes[0].set_xlabel('Latitude', fontsize=13)
axes[0].set_ylabel('Longitude', fontsize=13)
axes[0].set_title('Mosambik 2016', fontsize=subplot_title_size)
axes[0].set_aspect('equal')

moz.plot(ax=axes[1], color='white', edgecolor='grey')
urban_2017_df.sort_values(by='Prediction').plot(ax = axes[1], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)


axes[1].set_xlabel('Latitude', fontsize=13)
axes[1].set_title('Mosambik 2017', fontsize=subplot_title_size)

axes[1].set_aspect('equal')

moz.plot(ax=axes[2], color='white', edgecolor='grey')
urban_2019_df.sort_values(by='Prediction').plot(ax = axes[2], column='Prediction',
```

(continues on next page)

(continued from previous page)

```
                 cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[2].set_xlabel('Latitude', fontsize=13)
axes[2].set_title('Mosambik 2019', fontsize=subplot_title_size)
axes[2].set_aspect('equal')

moz.plot(ax=axes[3], color='white', edgecolor='grey')
urban_2020_df.sort_values(by='Prediction').plot(ax = axes[3], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[3].set_xlabel('Latitude', fontsize=13)
axes[3].set_title('Mosambik 2020', fontsize=subplot_title_size)
axes[3].set_aspect('equal')

moz.plot(ax=axes[4], color='white', edgecolor='grey')
urban_2021_df.sort_values(by='Prediction').plot(ax = axes[4], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[4].set_xlabel('Latitude', fontsize=13)
axes[4].set_title('Mosambik 2021', fontsize=subplot_title_size)
axes[4].set_aspect('equal')

sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=-4, vmax=4), cmap='coolwarm')
cbaxes = fig.add_axes([0.066, 0.2, .92, 0.03])
cbar = fig.colorbar(sm, orientation='horizontal',label='Asset Wealth',cax=cbaxes)


plt.show()
```



### Rural

#### Load Data for Mozambique 2016, 2017, 2019, 2020 and 2021

[41]:
```
model = keras.models.load_model('./resnet50_r_viirs.h5')
```

[42]:
```
moz_path_rural = '/mnt/datadisk/data/VIIRS/preprocessed/asset/rural/mozambique_2016_2021/
↪prep'

moz_rural = os.listdir(moz_path_rural)
len(moz_rural)
```

[42]:
```
1520
```

[43]:
```
moz_2016_rural = [i for i in moz_rural if '2016' in i]
moz_2017_rural = [i for i in moz_rural if '2017' in i]
moz_2019_rural = [i for i in moz_rural if '2019' in i]
moz_2020_rural = [i for i in moz_rural if '2020' in i]
moz_2021_rural = [i for i in moz_rural if '2021' in i]
```

[44]:
```
data_rural= {}
for moz_data_rural in [[moz_2016_rural,'2016'],[moz_2017_rural,'2017'],[moz_2019_rural,
↪'2019'],[moz_2020_rural,'2020'],[moz_2021_rural,'2021']]:
    data_rural[moz_data_rural[1]] = np.zeros(shape=(len(moz_data_rural[0]), 3, 1000,␣
↪1000))
    for index, img in tqdm(enumerate(moz_data_rural[0])):
        # Read in each Image
        with rasterio.open(os.path.join(moz_path_rural, img)) as i:
            array = i.read().astype("float32")

        # Ensure that the Array is not empty
        array[np.isnan(array)] = 0
        assert not np.any(np.isnan(array)), "Float"

        # Add to batch
        data_rural[moz_data_rural[1]][index] = array

        # Check if batch is already full (Note: Index in batch array is from 0...4 hence␣
↪we need to add +1 to batch_ele)
    data_rural[moz_data_rural[1]] = data_rural[moz_data_rural[1]].transpose(0, 2, 3, 1)
    assert len(data_rural[moz_data_rural[1]]) == len(moz_data_rural[0])
```

```
0it [00:00, ?it/s]
```

```
0it [00:00, ?it/s]
```

```
0it [00:00, ?it/s]
```

```
0it [00:00, ?it/s]
```

```
0it [00:00, ?it/s]
```

### Predict Asset Wealth

[45]:
```python
from tensorflow.keras import optimizers, models
preds_rural = {}
for year in tqdm(data_rural.keys()):
    preds_rural[year] = model.predict(data_rural[year])
print(preds_rural.keys())
len(preds_rural['2016'])
```

```
  0%|          | 0/5 [00:00<?, ?it/s]
```

```
dict_keys(['2016', '2017', '2019', '2020', '2021'])
```

[45]:
```
304
```

[46]:
```python
pred_rural_file_dict = {'Filename':
                        moz_2016_rural+
                        moz_2017_rural+
                        moz_2019_rural+
                        moz_2020_rural+
                        moz_2021_rural,
                        'Prediction':
                        [pred[0] for pred in preds_rural['2016']]+
                        [pred[0] for pred in preds_rural['2017']]+
                        [pred[0] for pred in preds_rural['2019']]+
                        [pred[0] for pred in preds_rural['2020']]+
                        [pred[0] for pred in preds_rural['2021']]}
pred_rural_file_df = pd.DataFrame(pred_rural_file_dict)
print(pred_rural_file_df.shape)
pred_rural_file_df.head(3)
```

```
(1520, 2)
```

[46]:

| | Filename | Prediction |
|---|---|---|
| 0 | -20.5253_34.0313_20160101-20161231_MOZ_r_10.0.tif | -0.588530 |
| 1 | -14.4227_38.3001_20160101-20161231_MOZ_r_10.0.tif | -0.469308 |
| 2 | -15.8257_38.5071_20160101-20161231_MOZ_r_10.0.tif | -0.588530 |

### Merge Predictions with Survey Data

[47]:
```python
asset_pred_rural_df = moz_df.merge(pred_rural_file_df)
print(asset_pred_rural_df.shape)
asset_pred_rural_df.head(3)
```

```
(1520, 6)
```

[47]:

| | URBAN_RURA | LATNUM | LONGNUM | SURVEY_YEAR | \ |
|---|---|---|---|---|---|
| 0 | 1 | -15.1365 | 36.5967 | 2016 | |
| 1 | 1 | -14.6769 | 36.4565 | 2016 | |
| 2 | 1 | -12.0948 | 34.8491 | 2016 | |

| | Filename | Prediction |
|---|---|---|
| 0 | -15.1365_36.5967_20160101-20161231_MOZ_r_10.0.tif | -0.266213 |
| 1 | -14.6769_36.4565_20160101-20161231_MOZ_r_10.0.tif | -0.588530 |
| 2 | -12.0948_34.8491_20160101-20161231_MOZ_r_10.0.tif | -0.588530 |

### Get Geocoordinates and create Geometry Objects

### Merge Predictions with Survey Data

[48]:
```
geometry = gpd.points_from_xy(asset_pred_rural_df.LONGNUM, asset_pred_rural_df.LATNUM)
```

[49]:
```
asset_pred_rural_df = gpd.GeoDataFrame(asset_pred_rural_df,
                        geometry=geometry,
                        crs=crs
                       )
asset_pred_rural_df = asset_pred_rural_df[['SURVEY_YEAR', 'geometry', 'Prediction']]
print(asset_pred_rural_df.shape)
asset_pred_rural_df.head(3)
```

```
(1520, 3)
```

[49]:

| | SURVEY_YEAR | geometry | Prediction |
|---|---|---|---|
| 0 | 2016 | POINT (36.59670 -15.13650) | -0.266213 |
| 1 | 2016 | POINT (36.45650 -14.67690) | -0.588530 |
| 2 | 2016 | POINT (34.84910 -12.09480) | -0.588530 |

[50]:
```
asset_pred_rural_df.to_csv('./moz_rural_asset_wealth_prediction.csv')
```

[51]:
```
rural_2016_df = asset_pred_rural_df[asset_pred_rural_df.SURVEY_YEAR==2016]
rural_2016_df.shape

rural_2017_df = asset_pred_rural_df[asset_pred_rural_df.SURVEY_YEAR==2017]
rural_2017_df.shape

rural_2019_df = asset_pred_rural_df[asset_pred_rural_df.SURVEY_YEAR==2019]
rural_2019_df.shape

rural_2020_df = asset_pred_rural_df[asset_pred_rural_df.SURVEY_YEAR==2020]
rural_2020_df.shape

rural_2021_df = asset_pred_rural_df[asset_pred_rural_df.SURVEY_YEAR==2021]
rural_2021_df.shape
```

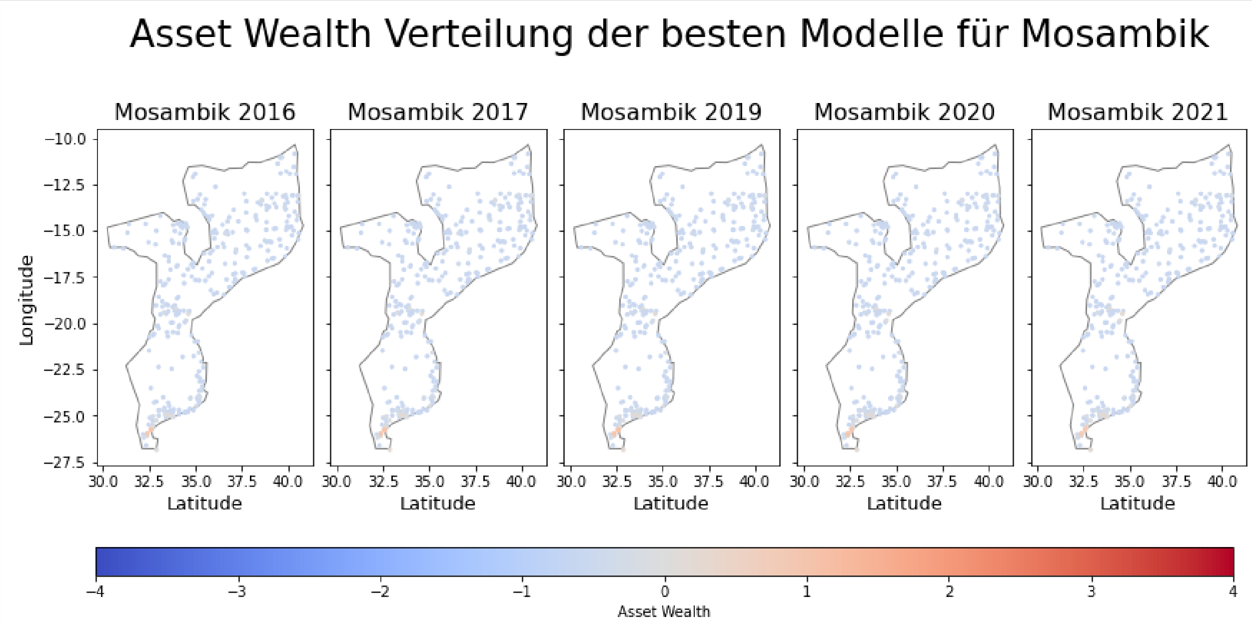
[51]:
```
(304, 3)
```

[52]:
```
fig, axes = plt.subplots(1, 5, sharey=True,constrained_layout=True, figsize=(12,9))


from matplotlib import colors

divnorm=colors.TwoSlopeNorm(vmin=-4, vcenter=0, vmax=4)
markersize=5
subplot_title_size = 16

axes[0].set_aspect('equal')
plt.suptitle('Asset Wealth Verteilung der besten Modelle für Mosambik', fontsize=26, y=.
↪8, x=0.53)
```

(continues on next page)

(continued from previous page)

```
moz.plot(ax=axes[0], color='white', edgecolor='grey')
rural_2016_df.sort_values(by='Prediction').plot(ax = axes[0], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)

axes[0].set_xlabel('Latitude', fontsize=13)
axes[0].set_ylabel('Longitude', fontsize=13)
axes[0].set_title('Mosambik 2016', fontsize=subplot_title_size)
axes[0].set_aspect('equal')

moz.plot(ax=axes[1], color='white', edgecolor='grey')
rural_2017_df.sort_values(by='Prediction').plot(ax = axes[1], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[1].set_xlabel('Latitude', fontsize=13)
axes[1].set_title('Mosambik 2017', fontsize=subplot_title_size)

axes[1].set_aspect('equal')

moz.plot(ax=axes[2], color='white', edgecolor='grey')
rural_2019_df.sort_values(by='Prediction').plot(ax = axes[2], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[2].set_xlabel('Latitude', fontsize=13)
axes[2].set_title('Mosambik 2019', fontsize=subplot_title_size)
axes[2].set_aspect('equal')

moz.plot(ax=axes[3], color='white', edgecolor='grey')
rural_2020_df.sort_values(by='Prediction').plot(ax = axes[3], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[3].set_xlabel('Latitude', fontsize=13)
axes[3].set_title('Mosambik 2020', fontsize=subplot_title_size)
axes[3].set_aspect('equal')

moz.plot(ax=axes[4], color='white', edgecolor='grey')
rural_2021_df.sort_values(by='Prediction').plot(ax = axes[4], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[4].set_xlabel('Latitude', fontsize=13)
axes[4].set_title('Mosambik 2021', fontsize=subplot_title_size)
axes[4].set_aspect('equal')

sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=-4, vmax=4), cmap='coolwarm')
cbaxes = fig.add_axes([0.066, 0.2, .92, 0.03])
cbar = fig.colorbar(sm, orientation='horizontal',label='Asset Wealth',cax=cbaxes)


plt.show()
```

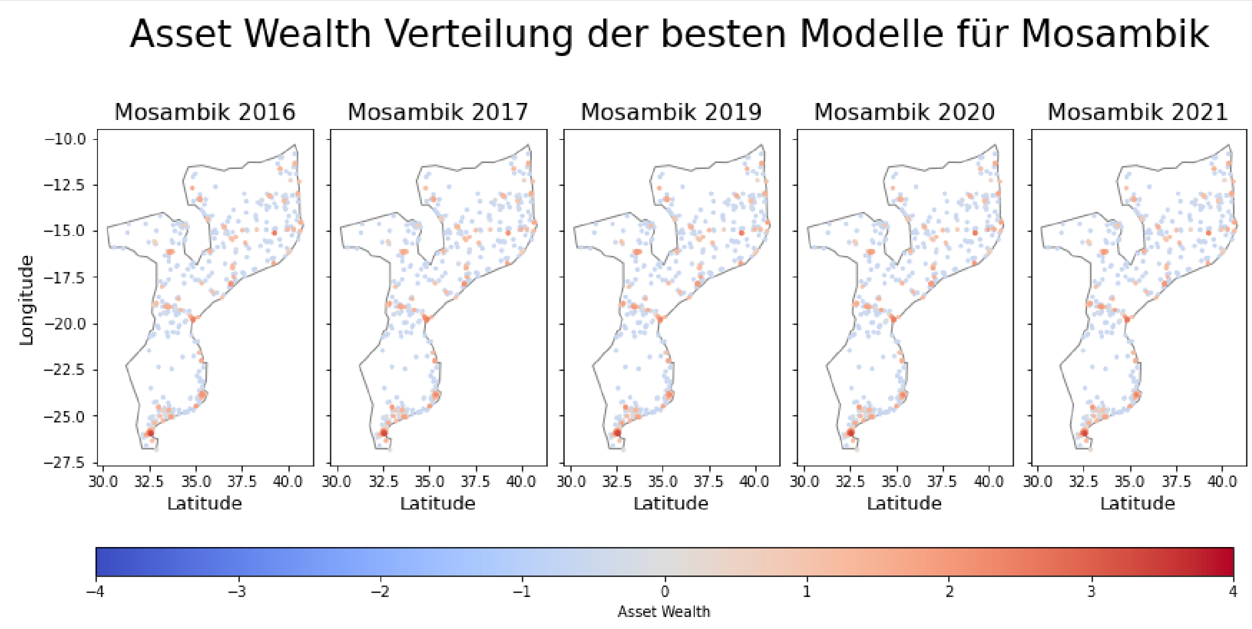



### Plot Predictions

[53]:
```
fig, axes = plt.subplots(1, 5, sharey=True,constrained_layout=True, figsize=(12,9))


from matplotlib import colors

divnorm=colors.TwoSlopeNorm(vmin=-4, vcenter=0, vmax=4)
markersize=5
subplot_title_size = 16

axes[0].set_aspect('equal')
plt.suptitle('Asset Wealth Verteilung der besten Modelle für Mosambik', fontsize=26, y=.
↪8, x=0.53)

moz.plot(ax=axes[0], color='white', edgecolor='grey')
rural_2016_df.sort_values(by='Prediction').plot(ax = axes[0], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)

urban_2016_df.sort_values(by='Prediction').plot(ax = axes[0], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)


axes[0].set_xlabel('Latitude', fontsize=13)
axes[0].set_ylabel('Longitude', fontsize=13)
axes[0].set_title('Mosambik 2016', fontsize=subplot_title_size)
axes[0].set_aspect('equal')

moz.plot(ax=axes[1], color='white', edgecolor='grey')
rural_2017_df.sort_values(by='Prediction').plot(ax = axes[1], column='Prediction',
                 cmap='coolwarm', norm=divnorm, markersize=markersize)
```
(continues on next page)

(continued from previous page)

```
urban_2017_df.sort_values(by='Prediction').plot(ax = axes[1], column='Prediction',
                cmap='coolwarm', norm=divnorm, markersize=markersize)


axes[1].set_xlabel('Latitude', fontsize=13)
axes[1].set_title('Mosambik 2017', fontsize=subplot_title_size)

axes[1].set_aspect('equal')

moz.plot(ax=axes[2], color='white', edgecolor='grey')
rural_2019_df.sort_values(by='Prediction').plot(ax = axes[2], column='Prediction',
                cmap='coolwarm', norm=divnorm, markersize=markersize)
urban_2019_df.sort_values(by='Prediction').plot(ax = axes[2], column='Prediction',
                cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[2].set_xlabel('Latitude', fontsize=13)
axes[2].set_title('Mosambik 2019', fontsize=subplot_title_size)
axes[2].set_aspect('equal')

moz.plot(ax=axes[3], color='white', edgecolor='grey')
rural_2020_df.sort_values(by='Prediction').plot(ax = axes[3], column='Prediction',
                cmap='coolwarm', norm=divnorm, markersize=markersize)
urban_2020_df.sort_values(by='Prediction').plot(ax = axes[3], column='Prediction',
                cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[3].set_xlabel('Latitude', fontsize=13)
axes[3].set_title('Mosambik 2020', fontsize=subplot_title_size)
axes[3].set_aspect('equal')

moz.plot(ax=axes[4], color='white', edgecolor='grey')
rural_2021_df.sort_values(by='Prediction').plot(ax = axes[4], column='Prediction',
                cmap='coolwarm', norm=divnorm, markersize=markersize)
urban_2021_df.sort_values(by='Prediction').plot(ax = axes[4], column='Prediction',
                cmap='coolwarm', norm=divnorm, markersize=markersize)
axes[4].set_xlabel('Latitude', fontsize=13)
axes[4].set_title('Mosambik 2021', fontsize=subplot_title_size)
axes[4].set_aspect('equal')

sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=-4, vmax=4), cmap='coolwarm')
cbaxes = fig.add_axes([0.066, 0.2, .92, 0.03])
cbar = fig.colorbar(sm, orientation='horizontal',label='Asset Wealth',cax=cbaxes)


plt.show()
```




[54]:
```
data = {'urban': {'2016': urban_2016_df,
                  '2017': urban_2017_df,
                  '2019': urban_2019_df,
                  '2020': urban_2020_df,
                  '2021': urban_2021_df},

        'rural': {'2016': rural_2016_df,
                  '2017': rural_2017_df,
                  '2019': rural_2019_df,
                  '2020': rural_2020_df,
                  '2021': rural_2021_df}
       }
statistics = []
for ur, ur_data in data.items():
    for year, pred in ur_data.items():
        statistics.append([ur,
                           year,
                           round(pred.Prediction.min(),2),
                           round(pred.Prediction.max(),2),
                           round(pred.Prediction.mean(),2),
                           round(pred.Prediction.median(),2),
                           round(pred.Prediction.var(),2),
                           round(pred.Prediction.skew(),2),
                           round(pred.Prediction.kurtosis(),2)])
pred_statistics_df = pd.DataFrame(data=statistics, columns=['Urban/Rural','Jahr','Min',
↪'Max', 'Arithmetisches\nMittel','Median','Varianz','Schiefe','Kurtosis'])
pred_statistics_df
```

```
['urban', '2016', 0.32, 3.38, 1.53, 1.45, 0.32, 0.57, 0.7]
['urban', '2017', 0.32, 3.3, 1.55, 1.51, 0.32, 0.47, 0.48]
['urban', '2019', 0.33, 3.36, 1.57, 1.52, 0.3, 0.61, 0.88]
['urban', '2020', 0.35, 3.4, 1.59, 1.53, 0.34, 0.58, 0.55]
```

(continues on next page)

(continued from previous page)

```
['urban', '2021', 0.33, 3.32, 1.57, 1.53, 0.34, 0.53, 0.47]
['rural', '2016', -0.62, 1.02, -0.52, -0.59, 0.04, 4.61, 25.61]
['rural', '2017', -0.63, 1.02, -0.51, -0.59, 0.05, 4.16, 20.34]
['rural', '2019', -0.62, 1.05, -0.51, -0.59, 0.05, 4.47, 23.02]
['rural', '2020', -0.67, 0.96, -0.51, -0.59, 0.05, 4.23, 20.31]
['rural', '2021', -0.59, 0.81, -0.5, -0.59, 0.05, 3.66, 14.73]
```

[54]:

```
  Urban/Rural  Jahr   Min   Max  Arithmetisches\nMittel  Median  Varianz  \
0       urban  2016  0.32  3.38                    1.53    1.45     0.32
1       urban  2017  0.32  3.30                    1.55    1.51     0.32
2       urban  2019  0.33  3.36                    1.57    1.52     0.30
3       urban  2020  0.35  3.40                    1.59    1.53     0.34
4       urban  2021  0.33  3.32                    1.57    1.53     0.34
5       rural  2016 -0.62  1.02                   -0.52   -0.59     0.04
6       rural  2017 -0.63  1.02                   -0.51   -0.59     0.05
7       rural  2019 -0.62  1.05                   -0.51   -0.59     0.05
8       rural  2020 -0.67  0.96                   -0.51   -0.59     0.05
9       rural  2021 -0.59  0.81                   -0.50   -0.59     0.05

   Schiefe   Kurtosis
0     0.57   0.700000
1     0.47   0.480000
2     0.61   0.880000
3     0.58   0.550000
4     0.53   0.470000
5     4.61  25.610001
6     4.16  20.340000
7     4.47  23.020000
8     4.23  20.309999
9     3.66  14.730000
```

### 4.2.3 Split and copy raw GeoTIFFs to separate directories for preprocessing

[1]:

```python
import os
import sys

sys.path.append("..")

from shutil import copyfile

from src.config import csv_path
from src.config import download_path_s2
from src.config import download_path_viirs
from src.config import countries

from src.data_utils import combine_wealth_dfs
from src.data_utils import get_label_for_img


sentinel_prep_path = '/mnt/datadisk/data/Sentinel2/preprocessed/asset/'
viirs_prep_path = '/mnt/datadisk/data/VIIRS/preprocessed/asset/'
```

### Sentinel-2

#### Separate Urban and Rural Files

[3]:
```
urban_files = []
rural_files = []
```

[4]:
```
dir_list = [directory for directory in os.listdir(download_path_s2) if os.path.isdir(os.
↪path.join(download_path_s2,directory)) and any(country in directory for country in␣
↪countries)]

for directory in dir_list:
    urban_files.extend([os.path.join(directory,file) for file in os.listdir(os.path.
↪join(download_path_s2,directory))
                        if file.endswith("u_2.0.tif")])
    rural_files.extend([os.path.join(directory, file) for file in os.listdir(os.path.
↪join(download_path_s2,directory))
                        if file.endswith("r_10.0.tif")])
```

#### Copy to preprocessing directory (separated by type of region)

[17]:
```
s2_urban_all_path = os.path.join(sentinel_prep_path, "urban", "all")
s2_rural_all_path = os.path.join(sentinel_prep_path, "rural", "all")
```

[17]:
```
'/mnt/datadisk/data/Sentinel2/preprocessed/asset/rural/all'
```

[ ]:
```
for file in urban_files:
    copyfile(os.path.join(download_path_s2,file), os.path.join(s2_urban_all_path, file))
for file in urban_files:
    copyfile(os.path.join(download_path_s2, file), os.path.join(s2_rural_all_path file))
```

#### Get Images for 2012-2014 and 2016-2020

[ ]:
```
wealth_df = combine_wealth(csv_path)
for filename in os.listdir(s2_urban_all_path):
    year = get_label_for_img(wealth_df, filename).SURVEY_YEAR
    if year < 2015:
        copyfile(os.path.join(s2_urban_all_path, filename), os.path.join(s2_rural_all_
↪path[:-3],'2012_2014',filename))
    elif year > 2015:
        copyfile(os.path.join(s2_rural_all_path, filename), os.path.join(s2_rural_all_
↪path[:-3],'2016_2020',filename))

for filename in os.listdir(s2_rural_all_path):
    year = get_label_for_img(wealth_df, filename).SURVEY_YEAR
    if year < 2015:
        copyfile(os.path.join(s2_rural_all_path, filename), os.path.join(s2_rural_all_
↪path[:-3],'2012_2014',filename))
    elif year > 2015:
```

(continues on next page)

(continued from previous page)

```
        copyfile(os.path.join(s2_rural_all_path, filename), os.path.join(s2_rural_all_
↪path[:-3],'2016_2020',filename))
```

### VIIRS

#### Separate Urban and Rural Files

[19]:
```
urban_files = []
rural_files = []
```

[20]:
```
dir_list = [directory for directory in os.listdir(download_path_viirs) if os.path.
↪isdir(os.path.join(download_path_viirs,directory)) and any(country in directory for␣
↪country in countries)]

for directory in dir_list:
    urban_files.extend([os.path.join(directory,file) for file in os.listdir(os.path.
↪join(download_path_viirs,directory))
                        if file.endswith("u_2.0.tif")])
    rural_files.extend([os.path.join(directory, file) for file in os.listdir(os.path.
↪join(download_path_viirs,directory))
                        if file.endswith("r_10.0.tif")])
```

#### Copy to preprocessing directory (separated by type of region)

[22]:
```
viirs_urban_all_path = os.path.join(viirs_prep_path, "urban", "all")
viirs_rural_all_path = os.path.join(viirs_prep_path, "rural", "all")
```

[ ]:
```
for file in urban_files:
    copyfile(os.path.join(download_path_viirs,file), os.path.join(viirs_urban_all_path,␣
↪file))
for file in urban_files:
    copyfile(os.path.join(download_path_viirs, file), os.path.join(viirs_rural_all_path␣
↪file))
```

#### Get Images for 2012-2014 and 2016-2020

[ ]:
```
for filename in os.listdir(viirs_urban_all_path):
    year = get_label_for_img(wealth_df, filename).SURVEY_YEAR
    if year < 2015:
        copyfile(os.path.join(viirs_urban_all_path, filename), os.path.join(viirs_urban_
↪all_path[:-3],'2012_2014',filename))
    elif year > 2015:
        copyfile(os.path.join(viirs_urban_all_path, filename), os.path.join(viirs_urban_
↪all_path[:-3],'2016_2020',filename))

for filename in os.listdir(s2_rural_all_path):
```

(continues on next page)

(continued from previous page)

```
    year = get_label_for_img(wealth_df, filename).SURVEY_YEAR
    if year < 2015:
        copyfile(os.path.join(s2_rural_all_path, filename), os.path.join(s2_rural_all_
↪path[:-3],'2012_2014',filename))
    elif year > 2015:
        copyfile(os.path.join(s2_rural_all_path, filename), os.path.join(s2_rural_all_
↪path[:-3],'2016_2020',filename))
```

# PYTHON MODULE INDEX

## S

# INDEX

## T



## V