Better Care and Feeding of Machine Learning Models

Towards reproducibility and transparency

Talk Structure

- Personal reproducibility disaster story
- What is reproducibility?
- Why does it matter?
- Python tools for reproducible workflows
  - Demo 1: cookiecutterML
  - Demo 2: push-button analysis
  - Demo 3: interpretability
- Downstream benefits : interpretability, evolvability

Industry Innovation aka.
Practical Machine Learning

Pushing boundaries of implementation rather than research

Me one year ago

- Working hard
- But unorganised folders
- Non-readable filenames
- No version control or README.md
- No tracking of Python libraries and environments used

collaborators not impressed

In fact my project folder turned into an entire workshop on reproducibility

How I felt...

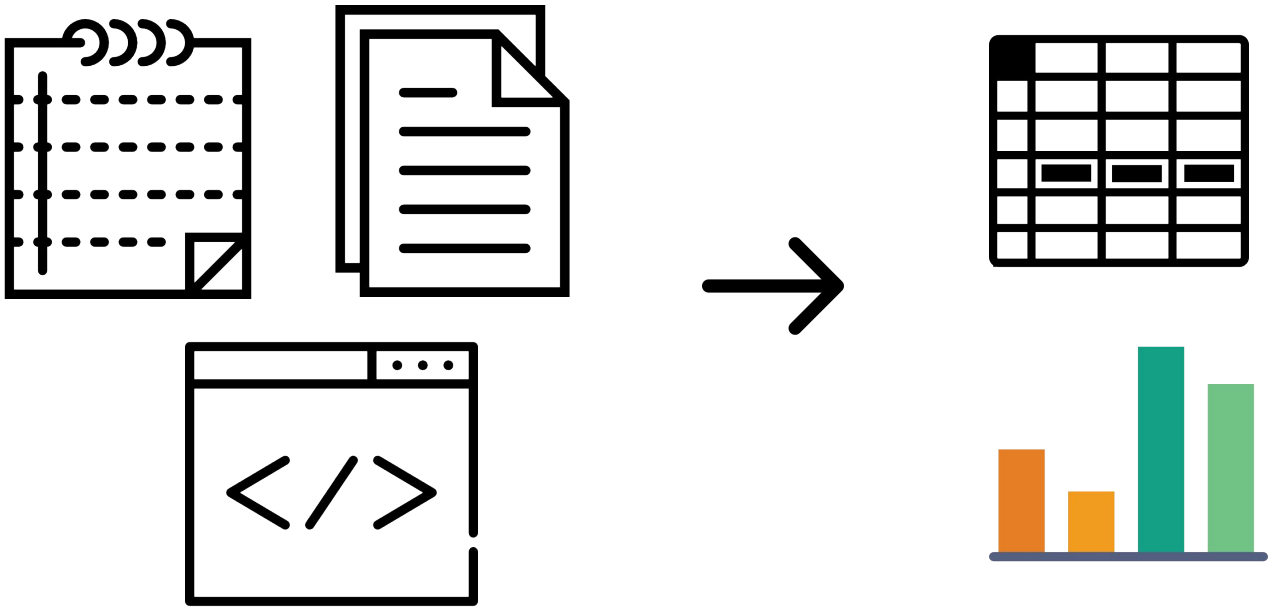
Future me also not impressed

Why reproducibility?

Your collaborators will thank you

Future self will thank you

Computational Reproducibility



https://www.practicereproducibleresearch.org/core-chapters/3-basic.html

Images: flaticon.com

March, 1978

Hedonic Housing Prices and the Demand for Clean Air

Daniel L. Rubinfeld
David Harrison, Jr., *Harvard University*

Reproducibility has different levels

complexity

Well-structured folders

Searchable filenames

Functions and automation with scripts and build tools

Reproducible Reports

Version control

Standardised environments

Containerisation

Some quick simple wins here.

Project templates = simple but a big win

Reproducible ML for Minimalists

Pared-down project template for reproducible Machine Learning. Adopted from Cookiecutter Data Science and Mario Krapp

Directory Structure

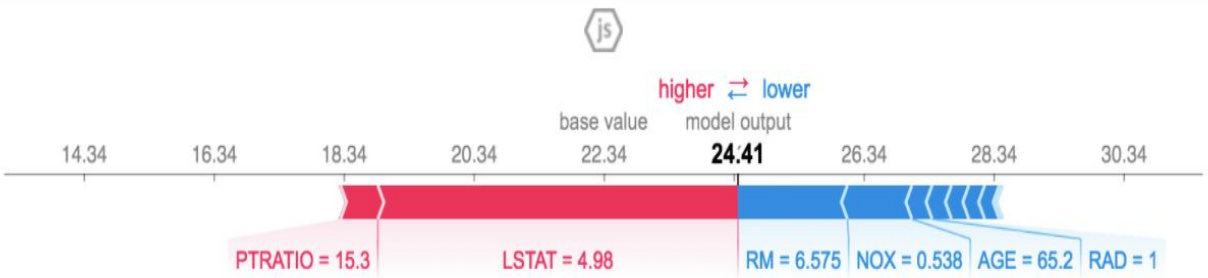
```
.
├── AUTHORS.md
├── LICENSE
├── README.md
├── models  <- compiled model .pkl or HDFS or .pb format
├── config  <- any configuration files
├── data
│   ├── interim <- data in intermediate processing stage
│   ├── processed <- data after all preprocessing has been done
│   └── raw <- original unmodified data acting as source of truth and provenance
├── docs  <- usage documentation or reference papers
├── notebooks <- jupyter notebooks for exploratory analysis and explanation
├── reports <- generated project artefacts eg. visualisations or tables
│   └── figures
└── src
    ├── data-proc <- scripts for processing data eg. transformations, dataset merges etc.
    ├── viz  <- scripts for visualisation during EDA, modelling, error analysis etc.
    ├── modeling    <- scripts for generating models
|--- environment.yml <- file with libraries and library versions for recreating the analysis environment
```

Push button analysis demo with housing data

delete all artefacts and recreate entire project using ONLY scripts!

Our own work - data diffs on top of code diffs

Benefits of Reproducible and Transparent models: Interpretability



The above explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in blue.

resources: https://github.com/slundberg/shap

Other benefits we are interested in

Containers == mix-and-match, manage and track models in the wild

Better Care and Feeding of Machine Learning Models

Towards reproducibility and transparency

How much would you pay for clean air?

a reproducible data science project with a great backstory to go with it

Hedonic Housing Prices and the Demand for Clean Air

Daniel L. Rubinfeld
David Harrison, Jr., *Harvard University*

Way back in 1978 Daniel and David published this paper. They asked how much more people were willing to pay for a house in an area with cleaner air. Knowing how uncomfortable things get when the haze hits, I can relate to why the authors were curious about this question

In [1]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import shap
import pandas as pd
import altair as alt
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.externals import joblib
```

read data in from `data/raw` folder

our raw data was downloaded with a shell script in `src/data` and exists in the `data/raw` folder. this data is **read only**. by keeping our sticky fingers off the raw data we make sure there is always an uncorrupted copy of the data we can fall back on.

In [2]:
```
# read data
df = pd.read_csv('../data/raw/housing.txt', index_col=0)
df.head()
```

Out[2]:

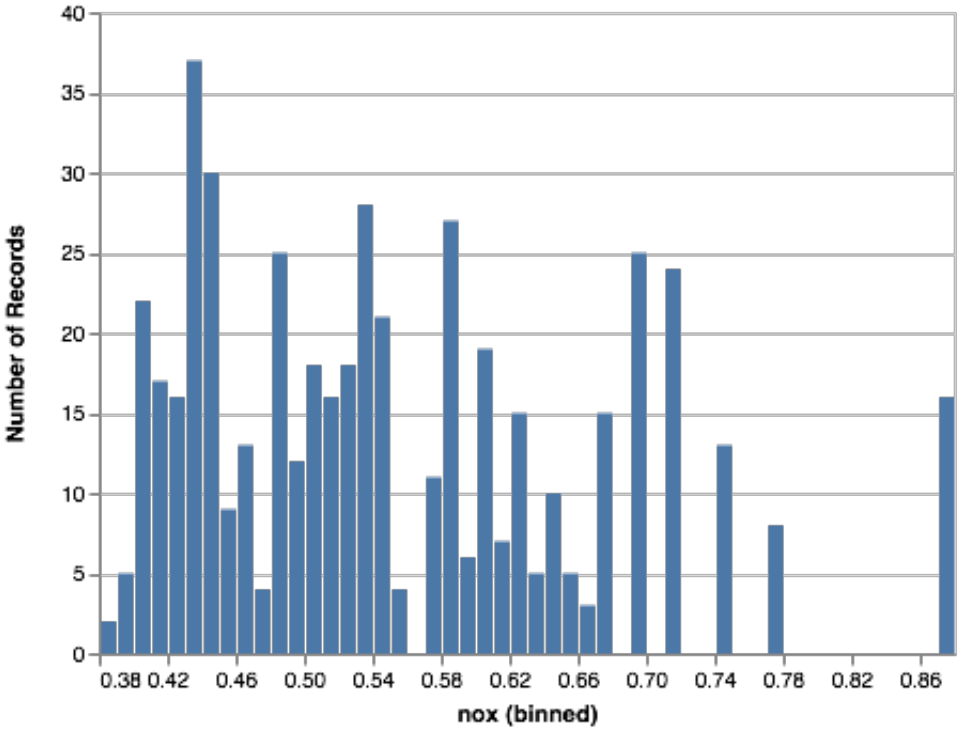
	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lst
1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.9
2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.1
3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.0
4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.9
5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.3

first things first - plot some distributions!

- data seems to be somewhat right-skewed, which could be a problem when we start modelling
- there are also some outliers worth digging into later

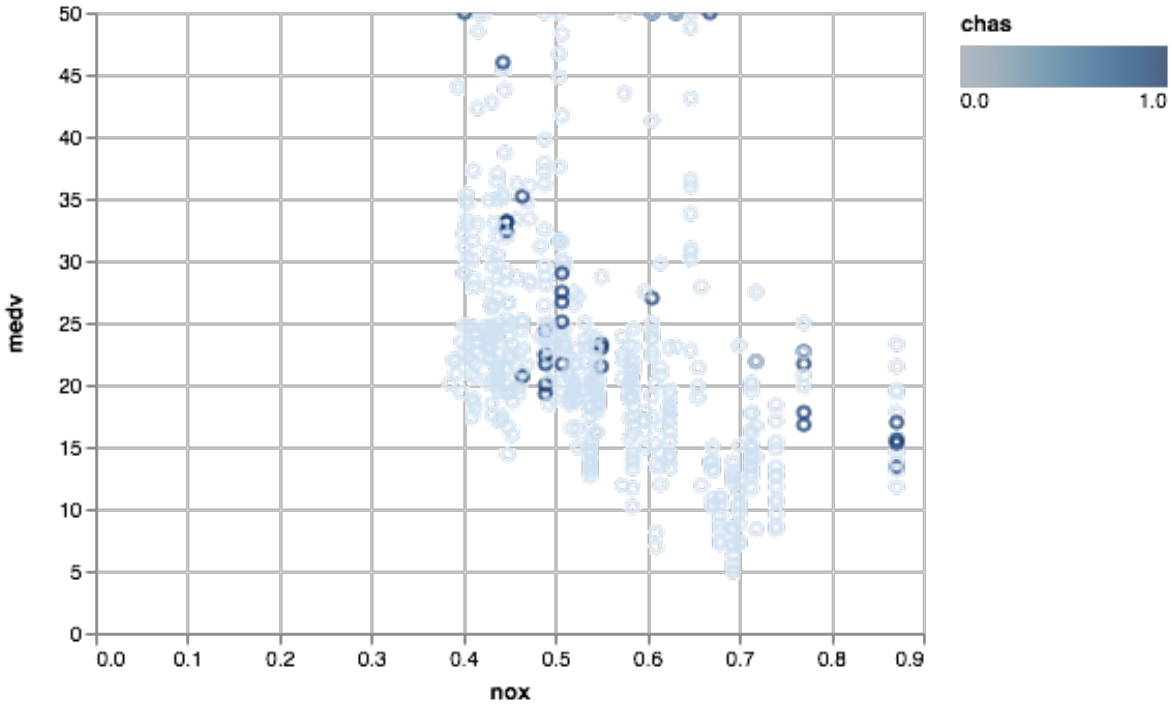
In [3]:
```
nox_chart = alt.Chart(df).mark_bar().encode(
    alt.X('nox', bin=alt.Bin(maxbins=70)),
    alt.Y('count()'))
nox_chart
```

Out[3]:



In [4]:
```
nox_medv_chart = alt.Chart(df).mark_point().encode(
    x='nox',
    y='medv', color='chas').interactive()
nox_medv_chart
```

Out[4]:



save our exploratory data analysis graphs into the `reports/figures` folder

- now we have a record of our initial analysis to refer to later
- future collaborators can also follow our train of thoguhts

In [5]:
```
nox_chart.save('../reports/figures/nox_chart.png')
nox_medv_chart.save('../reports/figures/nox_medv_chart.png')
```

split data into training and testing sets

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(*shap.datasets.
boston(), test_size=0.2, random_state=0)
```

In [7]:
```
X_train.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
220	0.35809	0.0	6.20	1.0	0.507	6.951	88.5	2.8617	8.0	307.0	17.4
71	0.15876	0.0	10.81	0.0	0.413	5.961	17.5	5.2873	4.0	305.0	19.2
240	0.11329	30.0	4.93	0.0	0.428	6.897	54.3	6.3361	6.0	300.0	16.6
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2
417	25.94060	0.0	18.10	0.0	0.679	5.304	89.1	1.6475	24.0	666.0	20.2

In [8]:
```
np.save('../data/processed/X_train', X_train)
np.save('../data/processed/X_test', X_test)
np.save('../data/processed/y_train', y_train)
np.save('../data/processed/y_test', y_test)
```

save train and test data in `data/processed` folder

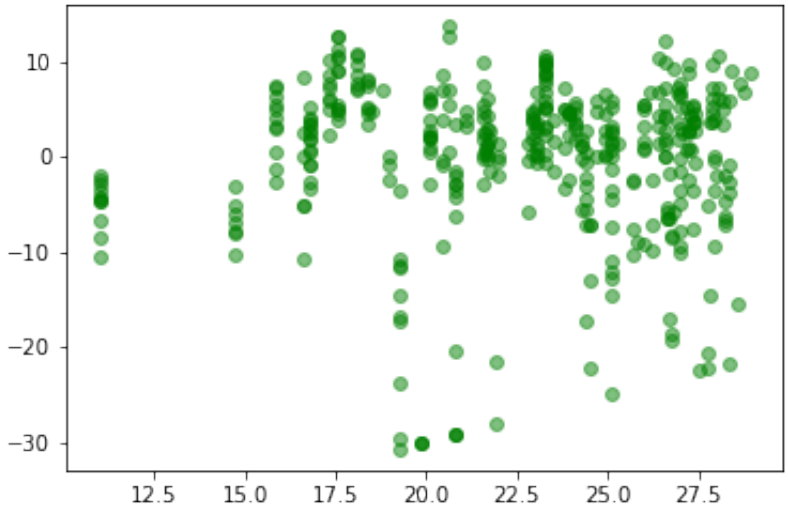
In [9]:
```
lm_one_feature = linear_model.LinearRegression()
```

In [10]:
```
# some feature engineering
# use only one feature
x_nox = X_train['NOX'].values.reshape(-1, 1)
```

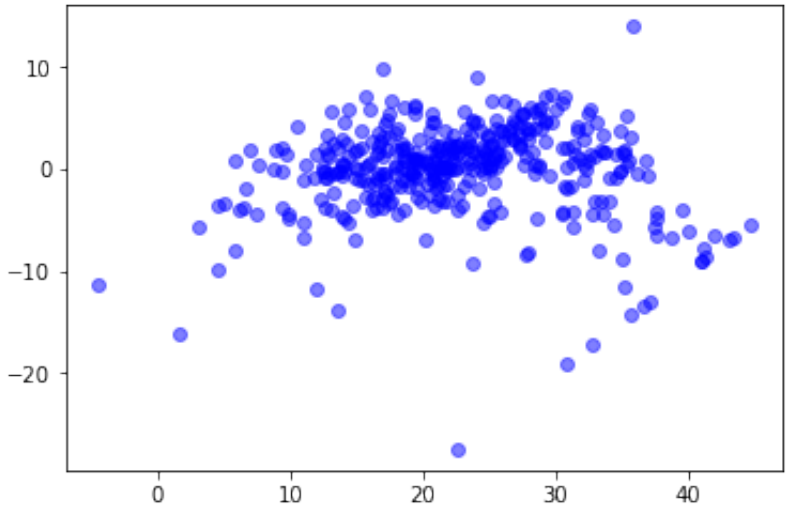
In [11]:
```
lm_one_feature.fit(x_nox, y_train)
```

Out[11]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normal
ize=False)
```

In [12]:
```
# plot residuals
plt.scatter(lm_one_feature.predict(x_nox), lm_one_feature.predict(x
_nox)-y_train, c='g', alpha=0.5)
plt.savefig('../reports/figures/lm_one_feature.jpg')
```



In [13]:
```
lm_all = linear_model.LinearRegression()
lm_all.fit(X_train, y_train)
plt.scatter(lm_all.predict(X_train), lm_all.predict(X_train)-y_trai
n, c='b', alpha=0.5)
plt.savefig('../reports/figures/lm_all.jpg')
```



save model

In [14]:
```
joblib.dump(lm_one_feature, '../models/lm_one_feature.pkl')
joblib.dump(lm_all, '../models/lm_all.pkl')
```

Out[14]: ['../models/lm_all.pkl']

model interpretability

In [15]:
```
shap.initjs()
```

In [16]:
```
model = RandomForestRegressor(n_estimators=1000, max_depth=4)
model.fit(X_train, y_train)
```

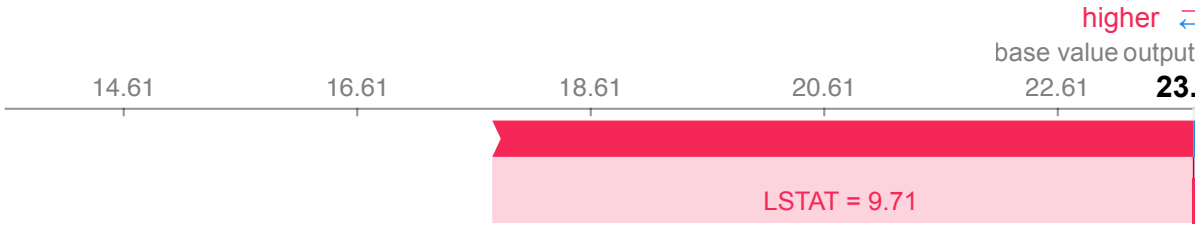
Out[16]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=4
,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs
=1,
           oob_score=False, random_state=None, verbose=0, warm_sta
rt=False)
```

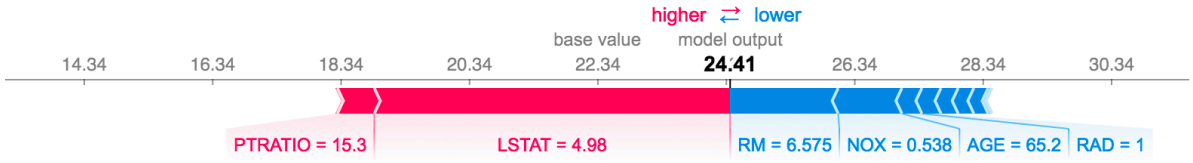
In [17]:
```
# explain the model's predictions using SHAP values
# (same syntax works for LightGBM, CatBoost, and scikit-learn model
s)
explainer = shap.KernelExplainer(model.predict, X_train)
shap_values = explainer.shap_values(X_test, nsamples=100)
```

```
100%|██████████| 102/102 [02:13<00:00,  1.30s/it]
```

In [18]:
```
# visualize the first prediction's explanation
shap.force_plot(shap_values[0,:], X_train.iloc[0,:])
```

Out[18]:



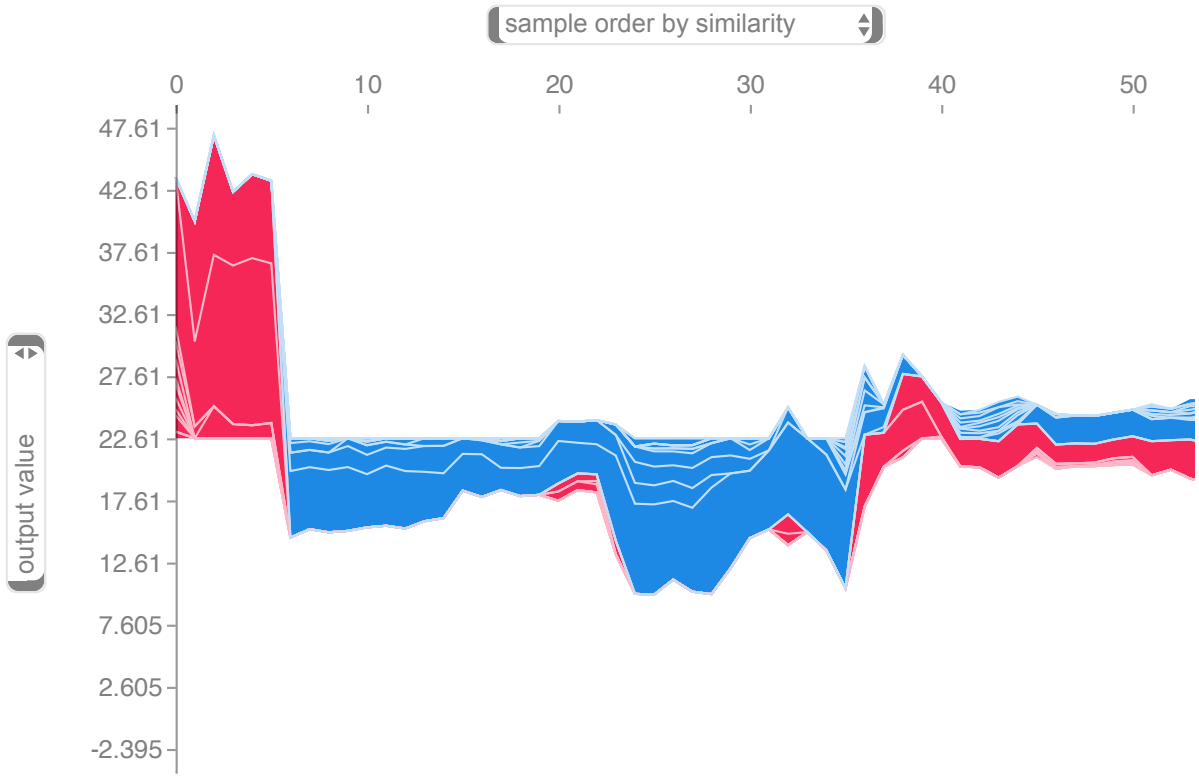


The above explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in blue.

If we take many explanations such as the one shown above, rotate them 90 degrees, and then stack them horizontally, we can see explanations for an entire dataset (in the notebook this plot is interactive):

In [19]:
```
#visualise training set predictions
shap.force_plot(shap_values, X_train)
```

Out[19]:



```
# visualize the training set predictions
shap.force_plot(shap_values, X)
```