Interpretable AI Project

##*Done by Saul Vassallo and David Scerri*

Below, when first run, will ask you to restart session. This is completely normal and will work on the second time running after session restarts

```
!pip install git+https://github.com/tensorflow/docs
!pip install pdpbox
!pip install alibi[tensorflow]
!pip install tf-explain
!pip install mmd_critic
!pip install lime
!pip install shap
```

```
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-
mx_2oh7l
  Running command git clone --filter=blob:none --quiet
https://github.com/tensorflow/docs /tmp/pip-req-build-mx_2oh7l
  Resolved https://github.com/tensorflow/docs to commit
c221d1e1af1ef5cc37c4a0879876f25f9cc1d981
  Preparing metadata (setup.py) ... ent already satisfied: astor in
/usr/local/lib/python3.11/dist-packages (from tensorflow-
docs==2024.11.18.43811) (0.8.1)
Requirement already satisfied: absl-py in
/usr/local/lib/python3.11/dist-packages (from tensorflow-
docs==2024.11.18.43811) (1.4.0)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from tensorflow-
docs==2024.11.18.43811) (3.1.5)
Requirement already satisfied: nbformat in
/usr/local/lib/python3.11/dist-packages (from tensorflow-
docs==2024.11.18.43811) (5.10.4)
Requirement already satisfied: protobuf>=3.12 in
/usr/local/lib/python3.11/dist-packages (from tensorflow-
docs==2024.11.18.43811) (4.25.5)
Requirement already satisfied: pyyaml in
/usr/local/lib/python3.11/dist-packages (from tensorflow-
docs==2024.11.18.43811) (6.0.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->tensorflow-
docs==2024.11.18.43811) (3.0.2)
Requirement already satisfied: fastjsonschema>=2.15 in
/usr/local/lib/python3.11/dist-packages (from nbformat->tensorflow-
docs==2024.11.18.43811) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in
```

```
/usr/local/lib/python3.11/dist-packages (from nbformat->tensorflow-
docs==2024.11.18.43811) (4.23.0)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in
/usr/local/lib/python3.11/dist-packages (from nbformat->tensorflow-
docs==2024.11.18.43811) (5.7.2)
Requirement already satisfied: traitlets>=5.1 in
/usr/local/lib/python3.11/dist-packages (from nbformat->tensorflow-
docs==2024.11.18.43811) (5.7.1)
Requirement already satisfied: attrs>=22.2.0 in
/usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6-
>nbformat->tensorflow-docs==2024.11.18.43811) (23.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6-
>nbformat->tensorflow-docs==2024.11.18.43811) (2024.10.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6-
>nbformat->tensorflow-docs==2024.11.18.43811) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in
/usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6-
>nbformat->tensorflow-docs==2024.11.18.43811) (0.22.3)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.11/dist-packages (from jupyter-core!
=5.0.*,>=4.12->nbformat->tensorflow-docs==2024.11.18.43811) (4.3.6)
Requirement already satisfied: pdpbox in
/usr/local/lib/python3.11/dist-packages (0.3.0)
Requirement already satisfied: joblib>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (1.4.2)
Requirement already satisfied: matplotlib>=3.6.2 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (3.10.0)
Requirement already satisfied: numpy>=1.21.5 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (1.26.4)
Requirement already satisfied: pandas>=1.4.4 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (2.2.2)
Requirement already satisfied: plotly>=5.9.0 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (5.24.1)
Requirement already satisfied: pqdm>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (0.2.0)
Requirement already satisfied: psutil>=5.9.0 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (5.9.5)
Requirement already satisfied: pytest in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (8.3.4)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (1.6.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (75.1.0)
Requirement already satisfied: sphinx>=5.0.2 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (8.1.3)
Requirement already satisfied: sphinx-rtd-theme>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (3.0.2)
```

```
Requirement already satisfied: tqdm>=4.64.1 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (4.67.1)
Requirement already satisfied: numpydoc>=1.4.0 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (1.8.0)
Requirement already satisfied: xgboost>=1.7.1 in
/usr/local/lib/python3.11/dist-packages (from pdpbox) (2.1.3)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6.2-
>pdpbox) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6.2-
>pdpbox) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6.2-
>pdpbox) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6.2-
>pdpbox) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6.2-
>pdpbox) (24.2)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6.2-
>pdpbox) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6.2-
>pdpbox) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6.2-
>pdpbox) (2.8.2)
Requirement already satisfied: tabulate>=0.8.10 in
/usr/local/lib/python3.11/dist-packages (from numpydoc>=1.4.0->pdpbox)
(0.9.0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.4.4->pdpbox)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.4.4->pdpbox)
(2024.2)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.11/dist-packages (from plotly>=5.9.0->pdpbox)
(9.0.0)
Requirement already satisfied: bounded-pool-executor in
/usr/local/lib/python3.11/dist-packages (from pqdm>=0.2.0->pdpbox)
(0.0.3)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from pqdm>=0.2.0->pdpbox)
(4.12.2)
Requirement already satisfied: scipy>=1.6.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.0.2-
>pdpbox) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.0.2-
>pdpbox) (3.5.0)
Requirement already satisfied: sphinxcontrib-applehelp>=1.0.7 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.0.0)
Requirement already satisfied: sphinxcontrib-devhelp>=1.0.6 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.0.0)
Requirement already satisfied: sphinxcontrib-htmlhelp>=2.0.6 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.1.0)
Requirement already satisfied: sphinxcontrib-jsmath>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(1.0.1)
Requirement already satisfied: sphinxcontrib-qthelp>=1.0.6 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.0.0)
Requirement already satisfied: sphinxcontrib-serializinghtml>=1.1.9 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.0.0)
Requirement already satisfied: Jinja2>=3.1 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(3.1.5)
Requirement already satisfied: Pygments>=2.17 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.18.0)
Requirement already satisfied: docutils<0.22,>=0.20 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(0.21.2)
Requirement already satisfied: snowballstemmer>=2.2 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.2.0)
Requirement already satisfied: babel>=2.13 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.16.0)
Requirement already satisfied: alabaster>=0.7.14 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(1.0.0)
Requirement already satisfied: imagesize>=1.3 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(1.4.1)
Requirement already satisfied: requests>=2.30.0 in
/usr/local/lib/python3.11/dist-packages (from sphinx>=5.0.2->pdpbox)
(2.32.3)
Requirement already satisfied: sphinxcontrib-jquery<5,>=4 in
/usr/local/lib/python3.11/dist-packages (from sphinx-rtd-theme>=1.1.1-
```

```
>pdpbox) (4.1)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.11/dist-packages (from xgboost>=1.7.1->pdpbox)
(2.21.5)
Requirement already satisfied: iniconfig in
/usr/local/lib/python3.11/dist-packages (from pytest->pdpbox) (2.0.0)
Requirement already satisfied: pluggy<2,>=1.5 in
/usr/local/lib/python3.11/dist-packages (from pytest->pdpbox) (1.5.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from Jinja2>=3.1-
>sphinx>=5.0.2->pdpbox) (3.0.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7-
>matplotlib>=3.6.2->pdpbox) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.30.0-
>sphinx>=5.0.2->pdpbox) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.30.0-
>sphinx>=5.0.2->pdpbox) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.30.0-
>sphinx>=5.0.2->pdpbox) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.30.0-
>sphinx>=5.0.2->pdpbox) (2024.12.14)
Requirement already satisfied: alibi[tensorflow] in
/usr/local/lib/python3.11/dist-packages (0.9.6)
Requirement already satisfied: numpy<2.0.0,>=1.16.2 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(1.26.4)
Requirement already satisfied: pandas<3.0.0,>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(2.2.2)
Requirement already satisfied: scikit-learn<2.0.0,>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(1.6.0)
Requirement already satisfied: spacy<4.0.0,>=2.0.0 in
/usr/local/lib/python3.11/dist-packages (from
spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.7.5)
Requirement already satisfied: blis<0.8.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(0.7.11)
Requirement already satisfied: scikit-image<0.23,>=0.17.2 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(0.22.0)
Requirement already satisfied: requests<3.0.0,>=2.21.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(2.32.3)
```

```
Requirement already satisfied: Pillow<11.0,>=5.4.1 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(10.4.0)
Requirement already satisfied: attrs<24.0.0,>=19.2.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(23.2.0)
Requirement already satisfied: scipy<2.0.0,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(1.13.1)
Requirement already satisfied: matplotlib<4.0.0,>=3.0.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(3.10.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(4.12.2)
Requirement already satisfied: dill<0.4.0,>=0.3.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(0.3.9)
Requirement already satisfied: transformers<5.0.0,>=4.7.0 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(4.47.1)
Requirement already satisfied: tqdm<5.0.0,>=4.28.1 in
/usr/local/lib/python3.11/dist-packages (from alibi[tensorflow])
(4.67.1)
Requirement already satisfied: tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0 in /usr/local/lib/python3.11/dist-packages
(from alibi[tensorflow]) (2.14.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib<4.0.0,>=3.0.0->alibi[tensorflow]) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib<4.0.0,>=3.0.0->alibi[tensorflow]) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib<4.0.0,>=3.0.0->alibi[tensorflow]) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib<4.0.0,>=3.0.0->alibi[tensorflow]) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib<4.0.0,>=3.0.0->alibi[tensorflow]) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib<4.0.0,>=3.0.0->alibi[tensorflow]) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib<4.0.0,>=3.0.0->alibi[tensorflow]) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
```

```
/usr/local/lib/python3.11/dist-packages (from pandas<3.0.0,>=1.0.0-
>alibi[tensorflow]) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas<3.0.0,>=1.0.0-
>alibi[tensorflow]) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.21.0-
>alibi[tensorflow]) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.21.0-
>alibi[tensorflow]) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.21.0-
>alibi[tensorflow]) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.21.0-
>alibi[tensorflow]) (2024.12.14)
Requirement already satisfied: networkx>=2.8 in
/usr/local/lib/python3.11/dist-packages (from scikit-
image<0.23,>=0.17.2->alibi[tensorflow]) (3.4.2)
Requirement already satisfied: imageio>=2.27 in
/usr/local/lib/python3.11/dist-packages (from scikit-
image<0.23,>=0.17.2->alibi[tensorflow]) (2.36.1)
Requirement already satisfied: tifffile>=2022.8.12 in
/usr/local/lib/python3.11/dist-packages (from scikit-
image<0.23,>=0.17.2->alibi[tensorflow]) (2024.12.12)
Requirement already satisfied: lazy_loader>=0.3 in
/usr/local/lib/python3.11/dist-packages (from scikit-
image<0.23,>=0.17.2->alibi[tensorflow]) (0.4)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-
learn<2.0.0,>=1.0.0->alibi[tensorflow]) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-
learn<2.0.0,>=1.0.0->alibi[tensorflow]) (3.5.0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.0.11)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.0.10)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
```

```
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (8.2.5)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.5.0)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (0.15.1)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.10.5)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.1.5)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (75.1.0)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in
/usr/local/lib/python3.11/dist-packages (from spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.5.0)
Requirement already satisfied: spacy-lookups-data<1.1.0,>=1.0.3 in
/usr/local/lib/python3.11/dist-packages (from
spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.0.5)
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (24.12.23)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1
in /usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.2.0)
```

```
Requirement already satisfied: h5py>=2.9.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (18.1.1)
Requirement already satisfied: ml-dtypes==0.2.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.2.0)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.4.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (4.25.5)
Requirement already satisfied: six>=1.12.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.37.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.69.0)
Requirement already satisfied: tensorboard<2.15,>=2.14 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0
in /usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.14.0)
Requirement already satisfied: keras<2.15,>=2.14.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.14.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from
transformers<5.0.0,>=4.7.0->alibi[tensorflow]) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in
/usr/local/lib/python3.11/dist-packages (from
transformers<5.0.0,>=4.7.0->alibi[tensorflow]) (0.27.1)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.11/dist-packages (from
transformers<5.0.0,>=4.7.0->alibi[tensorflow]) (6.0.2)
```

```
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.11/dist-packages (from
transformers<5.0.0,>=4.7.0->alibi[tensorflow]) (2024.11.6)
Requirement already satisfied: tokenizers<0.22,>=0.21 in
/usr/local/lib/python3.11/dist-packages (from
transformers<5.0.0,>=4.7.0->alibi[tensorflow]) (0.21.0)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.11/dist-packages (from
transformers<5.0.0,>=4.7.0->alibi[tensorflow]) (0.5.2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0-
>tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow])
(0.45.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.24.0->transformers<5.0.0,>=4.7.0->alibi[tensorflow])
(2024.10.0)
Requirement already satisfied: language-data>=1.2 in
/usr/local/lib/python3.11/dist-packages (from langcodes<4.0.0,>=3.2.0-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow])
(1.3.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!
=1.8.1,<3.0.0,>=1.7.4->spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (0.7.0)
Requirement already satisfied: pydantic-core==2.27.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!
=1.8.1,<3.0.0,>=1.7.4->spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.27.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.11/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow])
(2.27.0)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
/usr/local/lib/python3.11/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.11/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.11/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0-
>alibi[tensorflow]) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.1.3)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in
/usr/local/lib/python3.11/dist-packages (from thinc<8.3.0,>=8.2.2-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow])
```

```
(0.1.5)
Requirement already satisfied: click>=8.0.0 in
/usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow])
(8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in
/usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow])
(1.5.4)
Requirement already satisfied: rich>=10.11.0 in
/usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow])
(13.9.4)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in
/usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow])
(0.20.0)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in
/usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow])
(7.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow])
(3.0.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.11/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0-
>alibi[tensorflow]) (5.5.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.11/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0-
>alibi[tensorflow]) (0.4.1)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.11/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0-
>alibi[tensorflow]) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.11/dist-packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.3.1)
Requirement already satisfied: marisa-trie>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from language-data>=1.2-
>langcodes<4.0.0,>=3.2.0->spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.2.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich>=10.11.0-
>typer<1.0.0,>=0.3.0->spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from rich>=10.11.0-
>typer<1.0.0,>=0.3.0->spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0-
>rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy<4.0.0,>=2.0.0-
>spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (0.1.2)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.6.1)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.11/dist-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5-
>tensorboard<2.15,>=2.14->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0-
>alibi[tensorflow]) (3.2.2)
Collecting tf-explain
  Using cached tf_explain-0.3.1-py3-none-any.whl.metadata (9.3 kB)
Using cached tf_explain-0.3.1-py3-none-any.whl (43 kB)
Installing collected packages: tf-explain
Successfully installed tf-explain-0.3.1
Requirement already satisfied: mmd_critic in
/usr/local/lib/python3.11/dist-packages (0.1.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from mmd_critic) (1.26.4)
Requirement already satisfied: numexpr in
/usr/local/lib/python3.11/dist-packages (from mmd_critic) (2.10.2)
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 10.0 MB/s eta
0:00:00
etadata (setup.py) ... ent already satisfied: matplotlib in
/usr/local/lib/python3.11/dist-packages (from lime) (3.10.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from lime) (1.26.4)
Requirement already satisfied: scipy in
/usr/local/lib/python3.11/dist-packages (from lime) (1.13.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from lime) (4.67.1)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.11/dist-packages (from lime) (1.6.0)
Requirement already satisfied: scikit-image>=0.12 in
/usr/local/lib/python3.11/dist-packages (from lime) (0.22.0)
Requirement already satisfied: networkx>=2.8 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12-
>lime) (3.4.2)
Requirement already satisfied: pillow>=9.0.1 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12-
>lime) (10.4.0)
```

```
Requirement already satisfied: imageio>=2.27 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12-
>lime) (2.36.1)
Requirement already satisfied: tifffile>=2022.8.12 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12-
>lime) (2024.12.12)
Requirement already satisfied: packaging>=21 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12-
>lime) (24.2)
Requirement already satisfied: lazy_loader>=0.3 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12-
>lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.18-
>lime) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.18-
>lime) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime)
(4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime)
(1.4.8)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime)
(3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime)
(2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7-
>matplotlib->lime) (1.17.0)
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) ... e: filename=lime-0.2.0.1-py3-
none-any.whl size=283834
sha256=ab927e75e98e7d7fb377491d5d96172475036a37fc8186e18e0fe26972364b0
1
  Stored in directory:
/root/.cache/pip/wheels/85/fa/a3/9c2d44c9f3cd77cf4e533b58900b2bf4487f2
a17e8ec212a3d
Successfully built lime
Installing collected packages: lime
```

```
Successfully installed lime-0.2.0.1
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-
packages (0.46.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from shap) (1.26.4)
Requirement already satisfied: scipy in
/usr/local/lib/python3.11/dist-packages (from shap) (1.13.1)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.11/dist-packages (from shap) (1.6.0)
Requirement already satisfied: pandas in
/usr/local/lib/python3.11/dist-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in
/usr/local/lib/python3.11/dist-packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in
/usr/local/lib/python3.11/dist-packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in
/usr/local/lib/python3.11/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba in
/usr/local/lib/python3.11/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in
/usr/local/lib/python3.11/dist-packages (from shap) (3.1.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.11/dist-packages (from numba->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2024.2)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->shap)
(1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->shap)
(3.5.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas->shap) (1.17.0)
```

```
# TensorFlow
import tensorflow as tf
from tensorflow import keras
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import backend as K
```

```
from tf_explain.core.grad_cam import GradCAM

# Preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

# Criticism
from mmd_critic import MMDCritic
from mmd_critic.kernels import RBFKernel

# Commonly used modules
import numpy as np
import os
import cv2
import shap
from pdpbox import pdp
from time import time

# Images, plots, display, and visualization
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import IPython
from six.moves import urllib
from alibi.explainers import IntegratedGradients, Counterfactual
from alibi.utils.visualization import visualize_image_attr
from scipy.ndimage import gaussian_filter

from google.colab import drive

# Part 1 and 2
from sklearn.tree import DecisionTreeRegressor,
DecisionTreeClassifier, plot_tree
from sklearn.metrics import r2_score, accuracy_score
from lime import lime_tabular
from alibi.explainers import AnchorTabular

print(tf.__version__)
```

```
2.14.1
```

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
early_stop = EarlyStopping(monitor = 'val_loss', patience = 20,
restore_best_weights = True)
```

Cifar-10 Model

Function to unpickle the CIFAR-10 dataset files... taken from README.html

```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        data_dict = pickle.load(fo, encoding='bytes')
    return data_dict
```

Load CIFAR-10 Dataset and Preprocess

```
def load_cifar10(folder_path):
    x_train = []
    y_train = []
    for i in range(1, 6):
        batch = unpickle(os.path.join(folder_path, f"data_batch_{i}"))
        x_train.append(batch[b'data'])
        y_train.extend(batch[b'labels'])

    x_train = np.concatenate(x_train)
    y_train = np.array(y_train)

    test_batch = unpickle(os.path.join(folder_path, "test_batch"))
    x_test = test_batch[b'data']
    y_test = np.array(test_batch[b'labels'])

    # Reshape and normalize
    x_train = x_train.reshape(-1, 3, 32, 32).transpose(0, 2, 3,
1).astype('float32') / 255.0
    x_test = x_test.reshape(-1, 3, 32, 32).transpose(0, 2, 3,
1).astype('float32') / 255.0

    return (x_train, y_train), (x_test, y_test)

cifar10_folder = '/content/drive/My
Drive/InterpretableAI/Datasets/cifar-10/'
(x_train, y_train), (x_test, y_test) = load_cifar10(cifar10_folder)

y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

class_names = [
    "airplane", "automobile", "bird", "cat",
    "deer", "dog", "frog", "horse", "ship", "truck"
]
```

Define CNN Architecture

```
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

Compile CNN model

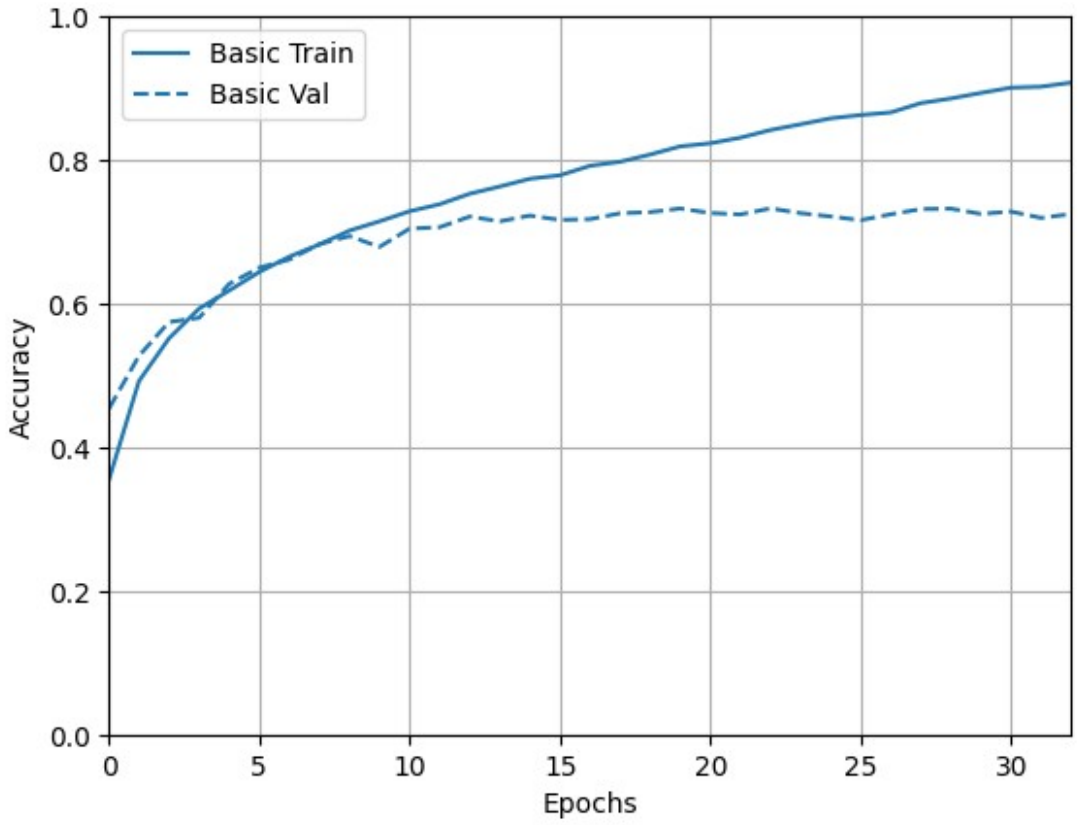
```
cnn_model.compile(optimizer = 'adam',
         loss = 'categorical_crossentropy',
         metrics = ['acc'])
```

Train the Model

```
history = cnn_model.fit(x_train, y_train, epochs = 500,
validation_split = 0.1, batch_size = 256, callbacks = [early_stop])

plotter = tfdocs.plots.HistoryPlotter(metric = 'acc')
plotter.plot({'Basic': history}, metric = 'acc')
plt.ylim([0, 1])
plt.ylabel('Accuracy')
plt.show()

Epoch 1/500
176/176 [==============================] - 65s 365ms/step - loss:
1.7625 - acc: 0.3545 - val_loss: 1.4842 - val_acc: 0.4536
Epoch 2/500
176/176 [==============================] - 63s 357ms/step - loss:
1.4157 - acc: 0.4920 - val_loss: 1.3258 - val_acc: 0.5272
Epoch 3/500
176/176 [==============================] - 54s 308ms/step - loss:
1.2651 - acc: 0.5516 - val_loss: 1.2121 - val_acc: 0.5744
Epoch 4/500
176/176 [==============================] - 55s 314ms/step - loss:
1.1602 - acc: 0.5930 - val_loss: 1.1811 - val_acc: 0.5802
Epoch 5/500
176/176 [==============================] - 53s 299ms/step - loss:
1.0890 - acc: 0.6182 - val_loss: 1.0619 - val_acc: 0.6274
Epoch 6/500
176/176 [==============================] - 54s 306ms/step - loss:
```

```
1.0200 - acc: 0.6439 - val_loss: 1.0154 - val_acc: 0.6500
Epoch 7/500
176/176 [==============================] - 52s 297ms/step - loss:
0.9607 - acc: 0.6653 - val_loss: 0.9916 - val_acc: 0.6614
Epoch 8/500
176/176 [==============================] - 50s 286ms/step - loss:
0.9117 - acc: 0.6824 - val_loss: 0.9328 - val_acc: 0.6828
Epoch 9/500
176/176 [==============================] - 51s 288ms/step - loss:
0.8595 - acc: 0.7016 - val_loss: 0.9152 - val_acc: 0.6940
Epoch 10/500
176/176 [==============================] - 52s 298ms/step - loss:
0.8235 - acc: 0.7146 - val_loss: 0.9639 - val_acc: 0.6790
Epoch 11/500
176/176 [==============================] - 51s 292ms/step - loss:
0.7868 - acc: 0.7286 - val_loss: 0.8572 - val_acc: 0.7044
Epoch 12/500
176/176 [==============================] - 50s 287ms/step - loss:
0.7565 - acc: 0.7384 - val_loss: 0.8817 - val_acc: 0.7064
Epoch 13/500
176/176 [==============================] - 52s 294ms/step - loss:
0.7189 - acc: 0.7530 - val_loss: 0.8342 - val_acc: 0.7212
Epoch 14/500
176/176 [==============================] - 50s 286ms/step - loss:
0.6888 - acc: 0.7630 - val_loss: 0.8571 - val_acc: 0.7146
Epoch 15/500
176/176 [==============================] - 51s 289ms/step - loss:
0.6548 - acc: 0.7738 - val_loss: 0.8395 - val_acc: 0.7226
Epoch 16/500
176/176 [==============================] - 50s 285ms/step - loss:
0.6384 - acc: 0.7787 - val_loss: 0.8501 - val_acc: 0.7166
Epoch 17/500
176/176 [==============================] - 52s 297ms/step - loss:
0.6019 - acc: 0.7919 - val_loss: 0.8764 - val_acc: 0.7176
Epoch 18/500
176/176 [==============================] - 50s 285ms/step - loss:
0.5824 - acc: 0.7975 - val_loss: 0.8477 - val_acc: 0.7258
Epoch 19/500
176/176 [==============================] - 51s 291ms/step - loss:
0.5534 - acc: 0.8076 - val_loss: 0.8399 - val_acc: 0.7274
Epoch 20/500
176/176 [==============================] - 51s 289ms/step - loss:
0.5230 - acc: 0.8190 - val_loss: 0.8469 - val_acc: 0.7324
Epoch 21/500
176/176 [==============================] - 53s 300ms/step - loss:
0.5108 - acc: 0.8233 - val_loss: 0.8600 - val_acc: 0.7262
Epoch 22/500
176/176 [==============================] - 51s 292ms/step - loss:
0.4923 - acc: 0.8310 - val_loss: 0.8805 - val_acc: 0.7242
```

```
Epoch 23/500
176/176 [==============================] - 51s 291ms/step - loss:
0.4561 - acc: 0.8417 - val_loss: 0.8541 - val_acc: 0.7326
Epoch 24/500
176/176 [==============================] - 52s 295ms/step - loss:
0.4384 - acc: 0.8497 - val_loss: 0.8978 - val_acc: 0.7260
Epoch 25/500
176/176 [==============================] - 52s 296ms/step - loss:
0.4115 - acc: 0.8580 - val_loss: 0.9100 - val_acc: 0.7214
Epoch 26/500
176/176 [==============================] - 49s 281ms/step - loss:
0.3998 - acc: 0.8626 - val_loss: 0.9701 - val_acc: 0.7162
Epoch 27/500
176/176 [==============================] - 50s 286ms/step - loss:
0.3830 - acc: 0.8662 - val_loss: 0.9626 - val_acc: 0.7244
Epoch 28/500
176/176 [==============================] - 53s 299ms/step - loss:
0.3504 - acc: 0.8792 - val_loss: 0.9594 - val_acc: 0.7316
Epoch 29/500
176/176 [==============================] - 53s 298ms/step - loss:
0.3319 - acc: 0.8856 - val_loss: 0.9516 - val_acc: 0.7322
Epoch 30/500
176/176 [==============================] - 50s 285ms/step - loss:
0.3116 - acc: 0.8934 - val_loss: 1.0088 - val_acc: 0.7250
Epoch 31/500
176/176 [==============================] - 50s 287ms/step - loss:
0.2903 - acc: 0.9007 - val_loss: 1.0255 - val_acc: 0.7282
Epoch 32/500
176/176 [==============================] - 51s 288ms/step - loss:
0.2830 - acc: 0.9021 - val_loss: 1.0493 - val_acc: 0.7194
Epoch 33/500
176/176 [==============================] - 52s 297ms/step - loss:
0.2683 - acc: 0.9078 - val_loss: 1.1025 - val_acc: 0.7248
```



Part 4: Interpretable AI for Computer Vision

1. Integrated Gradient

Initialise Integrated Gradients

```
ig = IntegratedGradients(cnn_model, n_steps=50,
method="gausslegendre", internal_batch_size=50)

image_index = 6
instance = np.expand_dims(x_test[image_index], axis=0)
```

Make prediction of image

```
predicted_probs = cnn_model.predict(instance)
predicted_label_index = np.argmax(predicted_probs, axis=1)[0]
predicted_label = class_names[predicted_label_index]
print(f"Predicted label for the instance: {predicted_label}")

1/1 [==============================] - 0s 152ms/step
Predicted label for the instance: automobile
```

Compute attributions for the prediction via explenation

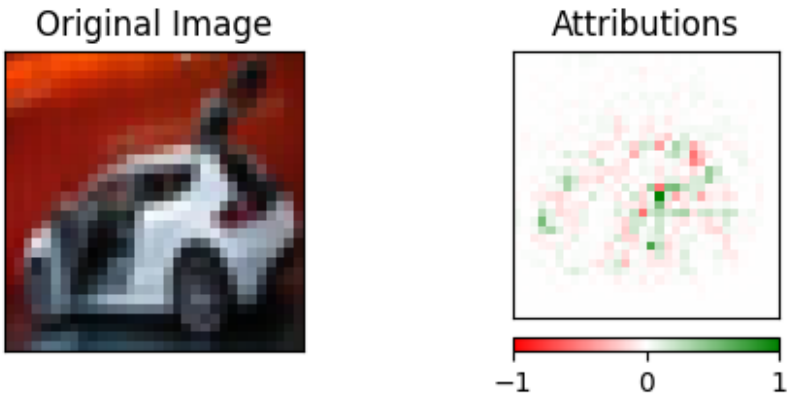
```
explanation = ig.explain(instance, baselines=None,
target=[predicted_label_index])
```

Extract attributions

```
attrs = explanation.attributions[0]
attrs_smoothed = gaussian_filter(attrs, sigma=0.5)
```

Visualise the original image and attributions

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(6, 2))

# Original Image
visualize_image_attr(attr=None, original_image=x_test[image_index],
method='original_image',
                     title='Original Image', plt_fig_axis=(fig,
ax[0]), use_pyplot=False)

# Attributions Heatmap
visualize_image_attr(attr=attrs.squeeze(),
original_image=x_test[image_index], method='heat_map',
                     sign='all', show_colorbar=True,
title='Attributions',
                     plt_fig_axis=(fig, ax[1]), use_pyplot=True)
plt.show()
```



The results of Integrated Gradients provide a visual representation of the features in the input image that contributed most significantly to the model's prediction. The attribution map indicates areas of the image where the gradients indicate that the model assigned a higher priority during categorisation. For example, in the the car image, the attributions are centred around specific structural components of the car, such as its perimeters and curves, which are essential for defining the object as a car. Positive attributes (green) highlight aspects that support the prediction, whereas negative attributes (red) show areas that contradict it. The distribution of attributions shows that the model focusses on unique and interpretable elements of the image, implying that its conclusions are somewhat consistent with human perception.

This insight is useful for ensuring that the model is using important features rather than false correlations.

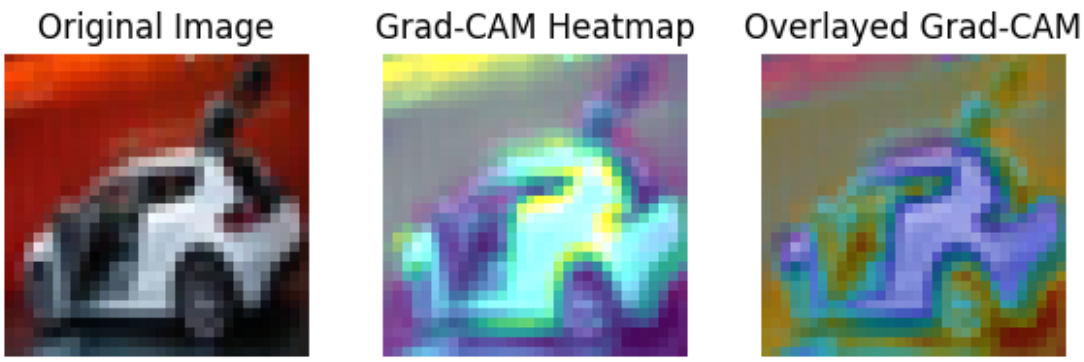
2. Grad-CAM

```
# Select an image from the test set
image = x_test[image_index]
label = np.argmax(y_test[image_index])

# Add a batch dimension to the image
image_with_batch = np.expand_dims(image, axis=0)

# Print the true label
print(f"True label index: {label}")
```

```
True label index: 1
```

```
# Initialize Grad-CAM
grad_cam = GradCAM()

# Define the target layer (adjust based on your CNN model's layer
names)
target_layer = 'conv2d'

# Compute Grad-CAM heatmap
explanation = grad_cam.explain(
    validation_data=(image_with_batch, None),
    model=cnn_model,
    class_index=label,  # Target class index
    layer_name=target_layer
)

# Normalize the heatmap
heatmap = explanation / np.max(explanation)

# Overlay the heatmap on the original image
overlayed_image = np.uint8(255 * image)  # Rescale original image to
0-255
heatmap_resized = np.uint8(255 * cv2.resize(heatmap, (32, 32)))  #
Resize heatmap to match image dimensions
overlayed_image = cv2.addWeighted(overlayed_image, 0.7,
cv2.applyColorMap(heatmap_resized, cv2.COLORMAP_JET), 0.4, 0)

# Plot the results
fig, ax = plt.subplots(1, 3, figsize=(6, 2))

# Original Image
ax[0].imshow(image)
ax[0].set_title("Original Image")
ax[0].axis('off')
```

```
# Heatmap
ax[1].imshow(heatmap, cmap='jet')
ax[1].set_title("Grad-CAM Heatmap")
ax[1].axis('off')

# Overlayed Grad-CAM
ax[2].imshow(overlayed_image)
ax[2].set_title("Overlayed Grad-CAM")
ax[2].axis('off')

plt.tight_layout()
plt.show()
```



The Grad-CAM results for the car image show where the convolutional neural network focusses during classification. Warm yellow and bright green regions on the heatmap represent areas of high importance, including the automobile's roof, rear brake lights and wheels, whereas cooler purple and blue regions show areas of lower relevance, such as the background and non-distinct aspects of the car. When the heatmap is placed on the original image, the highlighted regions blend in with the car's structural elements, indicating that the model's attention is focused on crucial aspects such as the car's shadow, roof and obvious outlines of its wheels. This visualisation indicates that the model based its predictions on meaningful car elements while ignoring irrelevant background data, hence enhancing the prediction process' reliability and interpretability.

Disable Eager mode for the rest of the Assignment

```
tf.compat.v1.disable_v2_behavior()
```

```
WARNING:tensorflow:From
/usr/local/lib/python3.11/dist-packages/tensorflow/python/compat/v2_co
mpat.py:108: disable_resource_variables (from
tensorflow.python.ops.variable_scope) is deprecated and will be
removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

Boston Housing Model

Load the data

```
(boston_train_features, boston_train_labels), (boston_test_features,
boston_test_labels) = keras.datasets.boston_housing.load_data()

print(boston_train_features.shape)
print(boston_test_features.shape)
print(boston_train_labels.shape)
print(boston_test_labels.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/boston_housing.npz
57026/57026 [==============================] - 0s 1us/step
(404, 13)
(102, 13)
(404,)
(102,)
```

Add the labels back to the dataset for feature engineering

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

```
boston_feature_names = [
  'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX',
  'RM', 'AGE', 'DIS', 'RAD', 'TAX',
  'PTRATIO', 'B', 'LSTAT'
]

boston_train_features_df = pd.DataFrame(boston_train_features, columns
= boston_feature_names)
boston_test_features_df = pd.DataFrame(boston_test_features, columns =
boston_feature_names)

print("Training Dataset Shape:", boston_train_features_df.shape)
print("\nFirst few rows of training data:")
print(boston_train_features_df.head())
print("\nDataset Info:")
print(boston_train_features_df.describe())
```

```
Training Dataset Shape: (404, 13)

First few rows of training data:
      CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS   RAD
TAX  \
0  1.23247   0.0   8.14   0.0  0.538  6.142   91.7  3.9769   4.0
307.0
1  0.02177  82.5   2.03   0.0  0.415  7.610   15.7  6.2700   2.0
348.0
2  4.89822   0.0  18.10   0.0  0.631  4.970  100.0  1.3325  24.0
666.0
3  0.03961   0.0   5.19   0.0  0.515  6.037   34.5  5.9853   5.0
224.0
4  3.69311   0.0  18.10   0.0  0.713  6.376   88.4  2.5671  24.0
666.0

   PTRATIO       B  LSTAT
0     21.0  396.90  18.72
1     14.7  395.38   3.11
2     20.2  375.52   3.26
3     20.2  396.90   8.01
4     20.2  391.43  14.65

Dataset Info:
             CRIM          ZN       INDUS        CHAS         NOX
RM  \
count  404.000000  404.000000  404.000000  404.000000  404.000000
404.000000
mean     3.745111   11.480198   11.104431    0.061881    0.557356
6.267082
std      9.240734   23.767711    6.811308    0.241238    0.117293
0.709788
min      0.006320    0.000000    0.460000    0.000000    0.385000
```

```
3.561000
25%      0.081437    0.000000    5.130000    0.000000    0.453000
5.874750
50%      0.268880    0.000000    9.690000    0.000000    0.538000
6.198500
75%      3.674808   12.500000   18.100000    0.000000    0.631000
6.609000
max     88.976200  100.000000   27.740000    1.000000    0.871000
8.725000

               AGE         DIS         RAD         TAX     PTRATIO
B  \
count  404.000000  404.000000  404.000000  404.000000  404.000000
404.000000
mean    69.010644    3.740271    9.440594  405.898515   18.475990
354.783168
std     27.940665    2.030215    8.698360  166.374543    2.200382
94.111148
min      2.900000    1.129600    1.000000  188.000000   12.600000
0.320000
25%     45.475000    2.077100    4.000000  279.000000   17.225000
374.672500
50%     78.500000    3.142300    5.000000  330.000000   19.100000
391.250000
75%     94.100000    5.118000   24.000000  666.000000   20.200000
396.157500
max    100.000000   10.710300   24.000000  711.000000   22.000000
396.900000

            LSTAT
count  404.000000
mean    12.740817
std      7.254545
min      1.730000
25%      6.890000
50%     11.395000
75%     17.092500
max     37.970000
```

Scale non categorical inputs (RAD) and outputs using Z-Score

```
def z_score_scale(data, columns_to_scale = None, exclude_columns =
None):
  """
  Scales columns using sklearn's StandardScaler. Can either specify
columns to scale
  or columns to exclude from scaling.

  Args:
```

```
    data: pandas DataFrame containing the data
    columns_to_scale: list of column names to scale (optional)
    exclude_columns: list of column names to exclude from scaling
(optional)

  Returns:
    DataFrame with scaled columns

  Note:
    If columns_to_scale is provided, exclude_columns is ignored.
    If neither is provided, all columns are scaled.
  """
  scaled_data = data.copy()

  # Determine which columns to scale
  if columns_to_scale is not None:
    # Use specified columns
    columns_to_process = columns_to_scale
  else:
    # Use all columns except excluded ones
    exclude_columns = exclude_columns or []
    columns_to_process = [col for col in data.columns if col not in
exclude_columns]

  scaler = StandardScaler()
  scaled_data[columns_to_process] =
scaler.fit_transform(data[columns_to_process])
  return scaled_data

boston_train_features_scaled_df =
z_score_scale(boston_train_features_df, exclude_columns = ['CHAS'])
boston_test_features_scaled_df =
z_score_scale(boston_test_features_df, exclude_columns = ['CHAS'])

scaler = StandardScaler()
boston_train_labels_scaled =
scaler.fit_transform(boston_train_labels.reshape(-1, 1))
boston_test_labels_scaled =
scaler.fit_transform(boston_test_labels.reshape(-1, 1))

boston_test_df = pd.concat([boston_test_features_scaled_df,
pd.DataFrame(boston_test_labels_scaled, columns = ['MEDV'])], axis =
1)

print(boston_train_features_scaled_df.head())
print(boston_test_features_scaled_df.head())
```

```
       CRIM        ZN     INDUS  CHAS       NOX        RM       AGE
DIS  \
0 -0.272246 -0.483615 -0.435762   0.0 -0.165227 -0.176443  0.813062
```

```
0.116698
1 -0.403427  2.991784 -1.333912   0.0 -1.215182  1.894346 -1.910361
1.247585
2  0.124940 -0.483615  1.028326   0.0  0.628642 -1.829688  1.110488 -
1.187439
3 -0.401494 -0.483615 -0.869402   0.0 -0.361560 -0.324558 -1.236672
1.107180
4 -0.005634 -0.483615  1.028326   0.0  1.328612  0.153642  0.694808 -
0.578572

        RAD       TAX   PTRATIO         B     LSTAT
0 -0.626249 -0.595170  1.148500  0.448077  0.825220
1 -0.856463 -0.348433 -1.718189  0.431906 -1.329202
2  1.675886  1.565287  0.784476  0.220617 -1.308500
3 -0.511142 -1.094663  0.784476  0.448077 -0.652926
4  1.675886  1.565287  0.784476  0.389882  0.263497
       CRIM        ZN     INDUS  CHAS       NOX        RM       AGE
DIS  \
0  2.804030 -0.507849  0.969609   0.0  1.231746  0.119341  1.147398 -
0.919353
1 -0.555306 -0.507849 -0.178017   0.0  0.025980 -0.659396  0.901658 -
0.700333
2 -0.568084 -0.507849 -0.861769   0.0 -0.266327 -0.551778 -0.742378
0.337890
3 -0.340188 -0.507849  1.179558   1.0  0.555786 -0.155683  0.891274 -
0.934680
4 -0.564990 -0.507849 -0.961070   0.0 -0.869210 -0.348499 -0.347809 -
0.111635

        RAD       TAX   PTRATIO         B     LSTAT
0  1.606093  1.407782  0.905130 -4.278295  2.513248
1 -0.455996  0.082144 -0.284858  0.390945  0.586043
2 -0.570556 -1.096201  0.905130  0.415707 -0.385064
3 -0.570556 -0.082144 -1.821927 -0.320553 -1.021462
4 -0.799677 -0.965903  0.062222  0.393484 -0.580186
```

```
boston_model = keras.Sequential([
  keras.layers.Dense(32, activation = 'relu', input_shape = (13,)),
  keras.layers.Dropout(0.1),
  keras.layers.Dense(1)
])

boston_model.compile(optimizer = 'adam',
           loss = 'mean_squared_error',
           metrics = ['mean_squared_error', 'mean_absolute_error'])

history = boston_model.fit(boston_train_features_scaled_df,
boston_train_labels_scaled, epochs = 500, validation_split = 0.1,
callbacks = [early_stop])
```

```
plotter = tfdocs.plots.HistoryPlotter(metric = 'mean_squared_error')
plotter.plot({'Basic': history}, metric = 'mean_squared_error')
plt.ylim([0, 1])
plt.ylabel('Mean Squared Error')
plt.show()
```

```
Epoch 1/500
12/12 [==============================] - 1s 35ms/step - loss: 0.9766 -
mean_squared_error: 0.9766 - mean_absolute_error: 0.7409 - val_loss:
0.4419 - val_mean_squared_error: 0.4419 - val_mean_absolute_error:
0.5220
Epoch 2/500
12/12 [==============================] - 0s 8ms/step - loss: 0.8104 -
mean_squared_error: 0.8104 - mean_absolute_error: 0.6342 - val_loss:
0.3032 - val_mean_squared_error: 0.3032 - val_mean_absolute_error:
0.4650
Epoch 3/500
12/12 [==============================] - 0s 9ms/step - loss: 0.6689 -
mean_squared_error: 0.6689 - mean_absolute_error: 0.5805 - val_loss:
0.2241 - val_mean_squared_error: 0.2241 - val_mean_absolute_error:
0.4009
Epoch 4/500
12/12 [==============================] - 0s 8ms/step - loss: 0.5688 -
mean_squared_error: 0.5688 - mean_absolute_error: 0.5111 - val_loss:
0.1757 - val_mean_squared_error: 0.1757 - val_mean_absolute_error:
0.3441
Epoch 5/500
12/12 [==============================] - 0s 7ms/step - loss: 0.4729 -
mean_squared_error: 0.4729 - mean_absolute_error: 0.4628 - val_loss:
0.1470 - val_mean_squared_error: 0.1470 - val_mean_absolute_error:
0.3110
Epoch 6/500
12/12 [==============================] - 0s 10ms/step - loss: 0.4367 -
mean_squared_error: 0.4367 - mean_absolute_error: 0.4512 - val_loss:
0.1277 - val_mean_squared_error: 0.1277 - val_mean_absolute_error:
0.2831
Epoch 7/500
12/12 [==============================] - 0s 9ms/step - loss: 0.4068 -
mean_squared_error: 0.4068 - mean_absolute_error: 0.4252 - val_loss:
0.1183 - val_mean_squared_error: 0.1183 - val_mean_absolute_error:
0.2725
Epoch 8/500
12/12 [==============================] - 0s 9ms/step - loss: 0.4052 -
mean_squared_error: 0.4052 - mean_absolute_error: 0.4296 - val_loss:
0.1181 - val_mean_squared_error: 0.1181 - val_mean_absolute_error:
0.2783
Epoch 9/500
12/12 [==============================] - 0s 9ms/step - loss: 0.3396 -
mean_squared_error: 0.3396 - mean_absolute_error: 0.3953 - val_loss:
0.1230 - val_mean_squared_error: 0.1230 - val_mean_absolute_error:
```

```
0.2871
Epoch 10/500
12/12 [==============================] - 0s 9ms/step - loss: 0.3376 -
mean_squared_error: 0.3376 - mean_absolute_error: 0.3845 - val_loss:
0.1228 - val_mean_squared_error: 0.1228 - val_mean_absolute_error:
0.2861
Epoch 11/500
12/12 [==============================] - 0s 11ms/step - loss: 0.3068 -
mean_squared_error: 0.3068 - mean_absolute_error: 0.3723 - val_loss:
0.1212 - val_mean_squared_error: 0.1212 - val_mean_absolute_error:
0.2832
Epoch 12/500
12/12 [==============================] - 0s 9ms/step - loss: 0.3060 -
mean_squared_error: 0.3060 - mean_absolute_error: 0.3864 - val_loss:
0.1185 - val_mean_squared_error: 0.1185 - val_mean_absolute_error:
0.2806
Epoch 13/500
12/12 [==============================] - 0s 7ms/step - loss: 0.3013 -
mean_squared_error: 0.3013 - mean_absolute_error: 0.3786 - val_loss:
0.1150 - val_mean_squared_error: 0.1150 - val_mean_absolute_error:
0.2763
Epoch 14/500
12/12 [==============================] - 0s 6ms/step - loss: 0.2934 -
mean_squared_error: 0.2934 - mean_absolute_error: 0.3662 - val_loss:
0.1163 - val_mean_squared_error: 0.1163 - val_mean_absolute_error:
0.2781
Epoch 15/500
12/12 [==============================] - 0s 7ms/step - loss: 0.2414 -
mean_squared_error: 0.2414 - mean_absolute_error: 0.3445 - val_loss:
0.1148 - val_mean_squared_error: 0.1148 - val_mean_absolute_error:
0.2766
Epoch 16/500
12/12 [==============================] - 0s 10ms/step - loss: 0.2458 -
mean_squared_error: 0.2458 - mean_absolute_error: 0.3453 - val_loss:
0.1118 - val_mean_squared_error: 0.1118 - val_mean_absolute_error:
0.2760
Epoch 17/500
12/12 [==============================] - 0s 7ms/step - loss: 0.2603 -
mean_squared_error: 0.2603 - mean_absolute_error: 0.3554 - val_loss:
0.1145 - val_mean_squared_error: 0.1145 - val_mean_absolute_error:
0.2790
Epoch 18/500
12/12 [==============================] - 0s 10ms/step - loss: 0.2647 -
mean_squared_error: 0.2647 - mean_absolute_error: 0.3631 - val_loss:
0.1201 - val_mean_squared_error: 0.1201 - val_mean_absolute_error:
0.2843
Epoch 19/500
12/12 [==============================] - 0s 10ms/step - loss: 0.2401 -
mean_squared_error: 0.2401 - mean_absolute_error: 0.3379 - val_loss:
```

```
0.1144 - val_mean_squared_error: 0.1144 - val_mean_absolute_error:
0.2776
Epoch 20/500
12/12 [==============================] - 0s 9ms/step - loss: 0.2448 -
mean_squared_error: 0.2448 - mean_absolute_error: 0.3600 - val_loss:
0.1144 - val_mean_squared_error: 0.1144 - val_mean_absolute_error:
0.2760
Epoch 21/500
12/12 [==============================] - 0s 13ms/step - loss: 0.2292 -
mean_squared_error: 0.2292 - mean_absolute_error: 0.3240 - val_loss:
0.1101 - val_mean_squared_error: 0.1101 - val_mean_absolute_error:
0.2712
Epoch 22/500
12/12 [==============================] - 0s 10ms/step - loss: 0.2338 -
mean_squared_error: 0.2338 - mean_absolute_error: 0.3357 - val_loss:
0.1075 - val_mean_squared_error: 0.1075 - val_mean_absolute_error:
0.2705
Epoch 23/500
12/12 [==============================] - 0s 8ms/step - loss: 0.2349 -
mean_squared_error: 0.2349 - mean_absolute_error: 0.3425 - val_loss:
0.1066 - val_mean_squared_error: 0.1066 - val_mean_absolute_error:
0.2714
Epoch 24/500
12/12 [==============================] - 0s 12ms/step - loss: 0.2029 -
mean_squared_error: 0.2029 - mean_absolute_error: 0.3223 - val_loss:
0.1021 - val_mean_squared_error: 0.1021 - val_mean_absolute_error:
0.2663
Epoch 25/500
12/12 [==============================] - 0s 10ms/step - loss: 0.2309 -
mean_squared_error: 0.2309 - mean_absolute_error: 0.3312 - val_loss:
0.1004 - val_mean_squared_error: 0.1004 - val_mean_absolute_error:
0.2648
Epoch 26/500
12/12 [==============================] - 0s 10ms/step - loss: 0.2481 -
mean_squared_error: 0.2481 - mean_absolute_error: 0.3322 - val_loss:
0.1004 - val_mean_squared_error: 0.1004 - val_mean_absolute_error:
0.2631
Epoch 27/500
12/12 [==============================] - 0s 9ms/step - loss: 0.2202 -
mean_squared_error: 0.2202 - mean_absolute_error: 0.3281 - val_loss:
0.1051 - val_mean_squared_error: 0.1051 - val_mean_absolute_error:
0.2674
Epoch 28/500
12/12 [==============================] - 0s 9ms/step - loss: 0.1946 -
mean_squared_error: 0.1946 - mean_absolute_error: 0.3021 - val_loss:
0.1005 - val_mean_squared_error: 0.1005 - val_mean_absolute_error:
0.2601
Epoch 29/500
12/12 [==============================] - 0s 9ms/step - loss: 0.2111 -
```

```
mean_squared_error: 0.2111 - mean_absolute_error: 0.3119 - val_loss:
0.1002 - val_mean_squared_error: 0.1002 - val_mean_absolute_error:
0.2615
Epoch 30/500
12/12 [==============================] - 0s 10ms/step - loss: 0.1914 -
mean_squared_error: 0.1914 - mean_absolute_error: 0.3115 - val_loss:
0.1010 - val_mean_squared_error: 0.1010 - val_mean_absolute_error:
0.2636
Epoch 31/500
12/12 [==============================] - 0s 11ms/step - loss: 0.2040 -
mean_squared_error: 0.2040 - mean_absolute_error: 0.3252 - val_loss:
0.0982 - val_mean_squared_error: 0.0982 - val_mean_absolute_error:
0.2591
Epoch 32/500
12/12 [==============================] - 0s 10ms/step - loss: 0.1900 -
mean_squared_error: 0.1900 - mean_absolute_error: 0.3059 - val_loss:
0.0932 - val_mean_squared_error: 0.0932 - val_mean_absolute_error:
0.2530
Epoch 33/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1930 -
mean_squared_error: 0.1930 - mean_absolute_error: 0.3042 - val_loss:
0.0885 - val_mean_squared_error: 0.0885 - val_mean_absolute_error:
0.2479
Epoch 34/500
12/12 [==============================] - 0s 10ms/step - loss: 0.2201 -
mean_squared_error: 0.2201 - mean_absolute_error: 0.3137 - val_loss:
0.0877 - val_mean_squared_error: 0.0877 - val_mean_absolute_error:
0.2477
Epoch 35/500
12/12 [==============================] - 0s 9ms/step - loss: 0.2123 -
mean_squared_error: 0.2123 - mean_absolute_error: 0.3332 - val_loss:
0.0932 - val_mean_squared_error: 0.0932 - val_mean_absolute_error:
0.2570
Epoch 36/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1941 -
mean_squared_error: 0.1941 - mean_absolute_error: 0.3046 - val_loss:
0.0958 - val_mean_squared_error: 0.0958 - val_mean_absolute_error:
0.2567
Epoch 37/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1875 -
mean_squared_error: 0.1875 - mean_absolute_error: 0.2931 - val_loss:
0.0901 - val_mean_squared_error: 0.0901 - val_mean_absolute_error:
0.2480
Epoch 38/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1846 -
mean_squared_error: 0.1846 - mean_absolute_error: 0.2943 - val_loss:
0.0883 - val_mean_squared_error: 0.0883 - val_mean_absolute_error:
0.2439
Epoch 39/500
```

```
12/12 [==============================] - 0s 7ms/step - loss: 0.1672 -
mean_squared_error: 0.1672 - mean_absolute_error: 0.2928 - val_loss:
0.0883 - val_mean_squared_error: 0.0883 - val_mean_absolute_error:
0.2451
Epoch 40/500
12/12 [==============================] - 0s 5ms/step - loss: 0.2153 -
mean_squared_error: 0.2153 - mean_absolute_error: 0.3236 - val_loss:
0.0946 - val_mean_squared_error: 0.0946 - val_mean_absolute_error:
0.2579
Epoch 41/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1921 -
mean_squared_error: 0.1921 - mean_absolute_error: 0.3033 - val_loss:
0.0865 - val_mean_squared_error: 0.0865 - val_mean_absolute_error:
0.2452
Epoch 42/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1860 -
mean_squared_error: 0.1860 - mean_absolute_error: 0.2999 - val_loss:
0.0816 - val_mean_squared_error: 0.0816 - val_mean_absolute_error:
0.2365
Epoch 43/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1604 -
mean_squared_error: 0.1604 - mean_absolute_error: 0.2808 - val_loss:
0.0822 - val_mean_squared_error: 0.0822 - val_mean_absolute_error:
0.2356
Epoch 44/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1560 -
mean_squared_error: 0.1560 - mean_absolute_error: 0.2737 - val_loss:
0.0832 - val_mean_squared_error: 0.0832 - val_mean_absolute_error:
0.2371
Epoch 45/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1664 -
mean_squared_error: 0.1664 - mean_absolute_error: 0.2879 - val_loss:
0.0857 - val_mean_squared_error: 0.0857 - val_mean_absolute_error:
0.2409
Epoch 46/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1620 -
mean_squared_error: 0.1620 - mean_absolute_error: 0.2760 - val_loss:
0.0869 - val_mean_squared_error: 0.0869 - val_mean_absolute_error:
0.2420
Epoch 47/500
12/12 [==============================] - 0s 9ms/step - loss: 0.1668 -
mean_squared_error: 0.1668 - mean_absolute_error: 0.2896 - val_loss:
0.0886 - val_mean_squared_error: 0.0886 - val_mean_absolute_error:
0.2447
Epoch 48/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1514 -
mean_squared_error: 0.1514 - mean_absolute_error: 0.2783 - val_loss:
0.0930 - val_mean_squared_error: 0.0930 - val_mean_absolute_error:
0.2535
```

```
Epoch 49/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1674 -
mean_squared_error: 0.1674 - mean_absolute_error: 0.2787 - val_loss:
0.0956 - val_mean_squared_error: 0.0956 - val_mean_absolute_error:
0.2580
Epoch 50/500
12/12 [==============================] - 0s 11ms/step - loss: 0.1571 -
mean_squared_error: 0.1571 - mean_absolute_error: 0.2795 - val_loss:
0.0907 - val_mean_squared_error: 0.0907 - val_mean_absolute_error:
0.2472
Epoch 51/500
12/12 [==============================] - 0s 9ms/step - loss: 0.1594 -
mean_squared_error: 0.1594 - mean_absolute_error: 0.2851 - val_loss:
0.0866 - val_mean_squared_error: 0.0866 - val_mean_absolute_error:
0.2416
Epoch 52/500
12/12 [==============================] - 0s 9ms/step - loss: 0.1494 -
mean_squared_error: 0.1494 - mean_absolute_error: 0.2821 - val_loss:
0.0893 - val_mean_squared_error: 0.0893 - val_mean_absolute_error:
0.2467
Epoch 53/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1483 -
mean_squared_error: 0.1483 - mean_absolute_error: 0.2763 - val_loss:
0.0878 - val_mean_squared_error: 0.0878 - val_mean_absolute_error:
0.2438
Epoch 54/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1529 -
mean_squared_error: 0.1529 - mean_absolute_error: 0.2723 - val_loss:
0.0829 - val_mean_squared_error: 0.0829 - val_mean_absolute_error:
0.2354
Epoch 55/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1705 -
mean_squared_error: 0.1705 - mean_absolute_error: 0.2909 - val_loss:
0.0854 - val_mean_squared_error: 0.0854 - val_mean_absolute_error:
0.2390
Epoch 56/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1498 -
mean_squared_error: 0.1498 - mean_absolute_error: 0.2754 - val_loss:
0.0873 - val_mean_squared_error: 0.0873 - val_mean_absolute_error:
0.2432
Epoch 57/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1623 -
mean_squared_error: 0.1623 - mean_absolute_error: 0.2877 - val_loss:
0.0821 - val_mean_squared_error: 0.0821 - val_mean_absolute_error:
0.2365
Epoch 58/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1650 -
mean_squared_error: 0.1650 - mean_absolute_error: 0.2794 - val_loss:
0.0816 - val_mean_squared_error: 0.0816 - val_mean_absolute_error:
```

```
0.2391
Epoch 59/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1776 -
mean_squared_error: 0.1776 - mean_absolute_error: 0.2798 - val_loss:
0.0758 - val_mean_squared_error: 0.0758 - val_mean_absolute_error:
0.2297
Epoch 60/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1656 -
mean_squared_error: 0.1656 - mean_absolute_error: 0.2814 - val_loss:
0.0744 - val_mean_squared_error: 0.0744 - val_mean_absolute_error:
0.2260
Epoch 61/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1340 -
mean_squared_error: 0.1340 - mean_absolute_error: 0.2489 - val_loss:
0.0741 - val_mean_squared_error: 0.0741 - val_mean_absolute_error:
0.2246
Epoch 62/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1463 -
mean_squared_error: 0.1463 - mean_absolute_error: 0.2641 - val_loss:
0.0765 - val_mean_squared_error: 0.0765 - val_mean_absolute_error:
0.2286
Epoch 63/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1354 -
mean_squared_error: 0.1354 - mean_absolute_error: 0.2630 - val_loss:
0.0787 - val_mean_squared_error: 0.0787 - val_mean_absolute_error:
0.2335
Epoch 64/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1582 -
mean_squared_error: 0.1582 - mean_absolute_error: 0.2760 - val_loss:
0.0790 - val_mean_squared_error: 0.0790 - val_mean_absolute_error:
0.2342
Epoch 65/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1406 -
mean_squared_error: 0.1406 - mean_absolute_error: 0.2733 - val_loss:
0.0769 - val_mean_squared_error: 0.0769 - val_mean_absolute_error:
0.2292
Epoch 66/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1574 -
mean_squared_error: 0.1574 - mean_absolute_error: 0.2746 - val_loss:
0.0759 - val_mean_squared_error: 0.0759 - val_mean_absolute_error:
0.2273
Epoch 67/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1278 -
mean_squared_error: 0.1278 - mean_absolute_error: 0.2500 - val_loss:
0.0755 - val_mean_squared_error: 0.0755 - val_mean_absolute_error:
0.2281
Epoch 68/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1459 -
mean_squared_error: 0.1459 - mean_absolute_error: 0.2693 - val_loss:
```

```
0.0741 - val_mean_squared_error: 0.0741 - val_mean_absolute_error:
0.2240
Epoch 69/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1407 -
mean_squared_error: 0.1407 - mean_absolute_error: 0.2595 - val_loss:
0.0747 - val_mean_squared_error: 0.0747 - val_mean_absolute_error:
0.2250
Epoch 70/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1518 -
mean_squared_error: 0.1518 - mean_absolute_error: 0.2746 - val_loss:
0.0760 - val_mean_squared_error: 0.0760 - val_mean_absolute_error:
0.2297
Epoch 71/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1381 -
mean_squared_error: 0.1381 - mean_absolute_error: 0.2644 - val_loss:
0.0728 - val_mean_squared_error: 0.0728 - val_mean_absolute_error:
0.2226
Epoch 72/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1492 -
mean_squared_error: 0.1492 - mean_absolute_error: 0.2626 - val_loss:
0.0711 - val_mean_squared_error: 0.0711 - val_mean_absolute_error:
0.2189
Epoch 73/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1338 -
mean_squared_error: 0.1338 - mean_absolute_error: 0.2548 - val_loss:
0.0727 - val_mean_squared_error: 0.0727 - val_mean_absolute_error:
0.2232
Epoch 74/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1451 -
mean_squared_error: 0.1451 - mean_absolute_error: 0.2694 - val_loss:
0.0733 - val_mean_squared_error: 0.0733 - val_mean_absolute_error:
0.2275
Epoch 75/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1395 -
mean_squared_error: 0.1395 - mean_absolute_error: 0.2708 - val_loss:
0.0746 - val_mean_squared_error: 0.0746 - val_mean_absolute_error:
0.2320
Epoch 76/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1393 -
mean_squared_error: 0.1393 - mean_absolute_error: 0.2680 - val_loss:
0.0748 - val_mean_squared_error: 0.0748 - val_mean_absolute_error:
0.2254
Epoch 77/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1188 -
mean_squared_error: 0.1188 - mean_absolute_error: 0.2447 - val_loss:
0.0731 - val_mean_squared_error: 0.0731 - val_mean_absolute_error:
0.2237
Epoch 78/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1320 -
```

```
mean_squared_error: 0.1320 - mean_absolute_error: 0.2582 - val_loss:
0.0744 - val_mean_squared_error: 0.0744 - val_mean_absolute_error:
0.2295
Epoch 79/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1271 -
mean_squared_error: 0.1271 - mean_absolute_error: 0.2505 - val_loss:
0.0741 - val_mean_squared_error: 0.0741 - val_mean_absolute_error:
0.2304
Epoch 80/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1216 -
mean_squared_error: 0.1216 - mean_absolute_error: 0.2446 - val_loss:
0.0713 - val_mean_squared_error: 0.0713 - val_mean_absolute_error:
0.2227
Epoch 81/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1278 -
mean_squared_error: 0.1278 - mean_absolute_error: 0.2504 - val_loss:
0.0729 - val_mean_squared_error: 0.0729 - val_mean_absolute_error:
0.2299
Epoch 82/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1283 -
mean_squared_error: 0.1283 - mean_absolute_error: 0.2568 - val_loss:
0.0731 - val_mean_squared_error: 0.0731 - val_mean_absolute_error:
0.2281
Epoch 83/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1087 -
mean_squared_error: 0.1087 - mean_absolute_error: 0.2314 - val_loss:
0.0745 - val_mean_squared_error: 0.0745 - val_mean_absolute_error:
0.2289
Epoch 84/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1306 -
mean_squared_error: 0.1306 - mean_absolute_error: 0.2538 - val_loss:
0.0745 - val_mean_squared_error: 0.0745 - val_mean_absolute_error:
0.2298
Epoch 85/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1360 -
mean_squared_error: 0.1360 - mean_absolute_error: 0.2467 - val_loss:
0.0706 - val_mean_squared_error: 0.0706 - val_mean_absolute_error:
0.2195
Epoch 86/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1178 -
mean_squared_error: 0.1178 - mean_absolute_error: 0.2417 - val_loss:
0.0702 - val_mean_squared_error: 0.0702 - val_mean_absolute_error:
0.2219
Epoch 87/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1388 -
mean_squared_error: 0.1388 - mean_absolute_error: 0.2537 - val_loss:
0.0706 - val_mean_squared_error: 0.0706 - val_mean_absolute_error:
0.2220
Epoch 88/500
```

```
12/12 [==============================] - 0s 6ms/step - loss: 0.1271 -
mean_squared_error: 0.1271 - mean_absolute_error: 0.2543 - val_loss:
0.0725 - val_mean_squared_error: 0.0725 - val_mean_absolute_error:
0.2279
Epoch 89/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1168 -
mean_squared_error: 0.1168 - mean_absolute_error: 0.2443 - val_loss:
0.0728 - val_mean_squared_error: 0.0728 - val_mean_absolute_error:
0.2311
Epoch 90/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1283 -
mean_squared_error: 0.1283 - mean_absolute_error: 0.2488 - val_loss:
0.0705 - val_mean_squared_error: 0.0705 - val_mean_absolute_error:
0.2272
Epoch 91/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1164 -
mean_squared_error: 0.1164 - mean_absolute_error: 0.2419 - val_loss:
0.0691 - val_mean_squared_error: 0.0691 - val_mean_absolute_error:
0.2205
Epoch 92/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1242 -
mean_squared_error: 0.1242 - mean_absolute_error: 0.2336 - val_loss:
0.0706 - val_mean_squared_error: 0.0706 - val_mean_absolute_error:
0.2223
Epoch 93/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1204 -
mean_squared_error: 0.1204 - mean_absolute_error: 0.2290 - val_loss:
0.0716 - val_mean_squared_error: 0.0716 - val_mean_absolute_error:
0.2271
Epoch 94/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1326 -
mean_squared_error: 0.1326 - mean_absolute_error: 0.2478 - val_loss:
0.0748 - val_mean_squared_error: 0.0748 - val_mean_absolute_error:
0.2331
Epoch 95/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1218 -
mean_squared_error: 0.1218 - mean_absolute_error: 0.2540 - val_loss:
0.0735 - val_mean_squared_error: 0.0735 - val_mean_absolute_error:
0.2289
Epoch 96/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1172 -
mean_squared_error: 0.1172 - mean_absolute_error: 0.2290 - val_loss:
0.0723 - val_mean_squared_error: 0.0723 - val_mean_absolute_error:
0.2271
Epoch 97/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1236 -
mean_squared_error: 0.1236 - mean_absolute_error: 0.2440 - val_loss:
0.0725 - val_mean_squared_error: 0.0725 - val_mean_absolute_error:
0.2285
```

```
Epoch 98/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1250 -
mean_squared_error: 0.1250 - mean_absolute_error: 0.2527 - val_loss:
0.0745 - val_mean_squared_error: 0.0745 - val_mean_absolute_error:
0.2328
Epoch 99/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1075 -
mean_squared_error: 0.1075 - mean_absolute_error: 0.2389 - val_loss:
0.0730 - val_mean_squared_error: 0.0730 - val_mean_absolute_error:
0.2292
Epoch 100/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1239 -
mean_squared_error: 0.1239 - mean_absolute_error: 0.2394 - val_loss:
0.0710 - val_mean_squared_error: 0.0710 - val_mean_absolute_error:
0.2238
Epoch 101/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1140 -
mean_squared_error: 0.1140 - mean_absolute_error: 0.2409 - val_loss:
0.0735 - val_mean_squared_error: 0.0735 - val_mean_absolute_error:
0.2325
Epoch 102/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1062 -
mean_squared_error: 0.1062 - mean_absolute_error: 0.2283 - val_loss:
0.0706 - val_mean_squared_error: 0.0706 - val_mean_absolute_error:
0.2229
Epoch 103/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1198 -
mean_squared_error: 0.1198 - mean_absolute_error: 0.2432 - val_loss:
0.0725 - val_mean_squared_error: 0.0725 - val_mean_absolute_error:
0.2263
Epoch 104/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1210 -
mean_squared_error: 0.1210 - mean_absolute_error: 0.2384 - val_loss:
0.0774 - val_mean_squared_error: 0.0774 - val_mean_absolute_error:
0.2380
Epoch 105/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1138 -
mean_squared_error: 0.1138 - mean_absolute_error: 0.2360 - val_loss:
0.0728 - val_mean_squared_error: 0.0728 - val_mean_absolute_error:
0.2258
Epoch 106/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1239 -
mean_squared_error: 0.1239 - mean_absolute_error: 0.2434 - val_loss:
0.0732 - val_mean_squared_error: 0.0732 - val_mean_absolute_error:
0.2299
Epoch 107/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1163 -
mean_squared_error: 0.1163 - mean_absolute_error: 0.2387 - val_loss:
0.0721 - val_mean_squared_error: 0.0721 - val_mean_absolute_error:
0.2319
```

```
Epoch 108/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1235 -
mean_squared_error: 0.1235 - mean_absolute_error: 0.2412 - val_loss:
0.0705 - val_mean_squared_error: 0.0705 - val_mean_absolute_error:
0.2281
Epoch 109/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1219 -
mean_squared_error: 0.1219 - mean_absolute_error: 0.2358 - val_loss:
0.0704 - val_mean_squared_error: 0.0704 - val_mean_absolute_error:
0.2271
Epoch 110/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1207 -
mean_squared_error: 0.1207 - mean_absolute_error: 0.2442 - val_loss:
0.0680 - val_mean_squared_error: 0.0680 - val_mean_absolute_error:
0.2202
Epoch 111/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1242 -
mean_squared_error: 0.1242 - mean_absolute_error: 0.2433 - val_loss:
0.0690 - val_mean_squared_error: 0.0690 - val_mean_absolute_error:
0.2239
Epoch 112/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1173 -
mean_squared_error: 0.1173 - mean_absolute_error: 0.2427 - val_loss:
0.0691 - val_mean_squared_error: 0.0691 - val_mean_absolute_error:
0.2244
Epoch 113/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1029 -
mean_squared_error: 0.1029 - mean_absolute_error: 0.2287 - val_loss:
0.0688 - val_mean_squared_error: 0.0688 - val_mean_absolute_error:
0.2237
Epoch 114/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1135 -
mean_squared_error: 0.1135 - mean_absolute_error: 0.2310 - val_loss:
0.0687 - val_mean_squared_error: 0.0687 - val_mean_absolute_error:
0.2265
Epoch 115/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1084 -
mean_squared_error: 0.1084 - mean_absolute_error: 0.2318 - val_loss:
0.0687 - val_mean_squared_error: 0.0687 - val_mean_absolute_error:
0.2274
Epoch 116/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1116 -
mean_squared_error: 0.1116 - mean_absolute_error: 0.2265 - val_loss:
0.0660 - val_mean_squared_error: 0.0660 - val_mean_absolute_error:
0.2183
Epoch 117/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1089 -
mean_squared_error: 0.1089 - mean_absolute_error: 0.2329 - val_loss:
0.0680 - val_mean_squared_error: 0.0680 - val_mean_absolute_error:
```

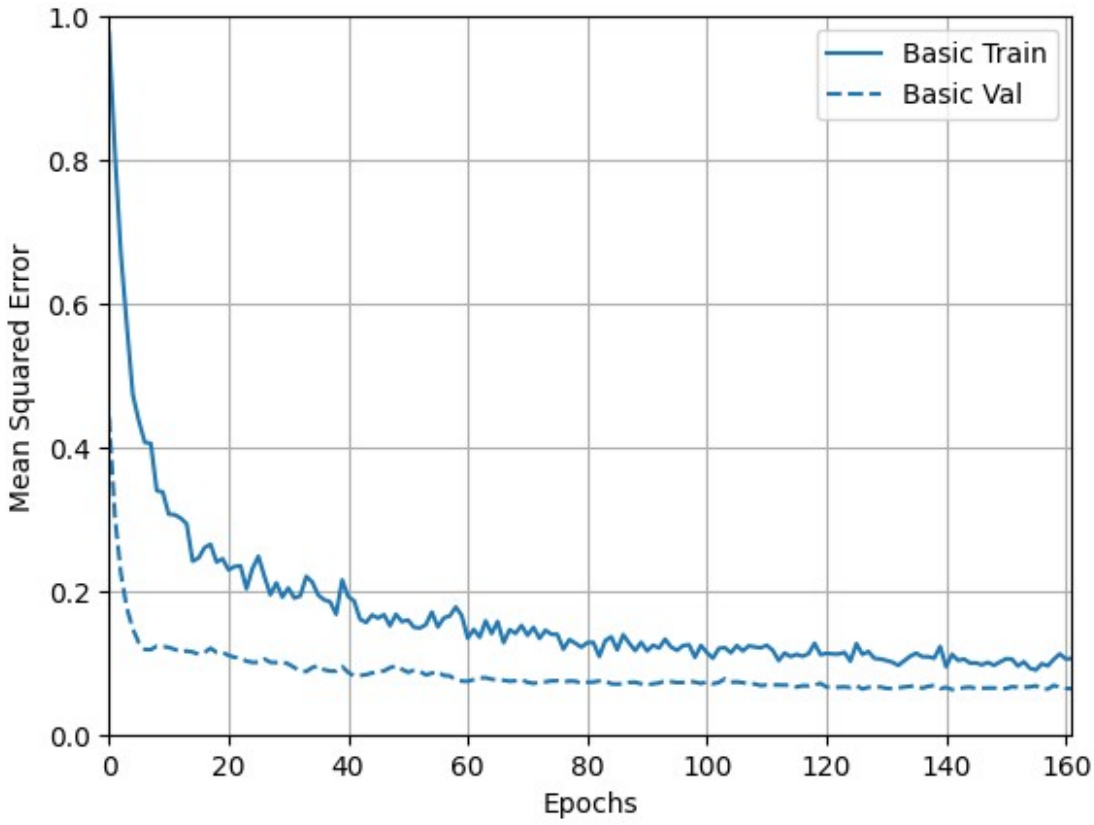
```
0.2241
Epoch 118/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1125 -
mean_squared_error: 0.1125 - mean_absolute_error: 0.2365 - val_loss:
0.0677 - val_mean_squared_error: 0.0677 - val_mean_absolute_error:
0.2216
Epoch 119/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1269 -
mean_squared_error: 0.1269 - mean_absolute_error: 0.2472 - val_loss:
0.0692 - val_mean_squared_error: 0.0692 - val_mean_absolute_error:
0.2252
Epoch 120/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1112 -
mean_squared_error: 0.1112 - mean_absolute_error: 0.2341 - val_loss:
0.0711 - val_mean_squared_error: 0.0711 - val_mean_absolute_error:
0.2302
Epoch 121/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1129 -
mean_squared_error: 0.1129 - mean_absolute_error: 0.2247 - val_loss:
0.0661 - val_mean_squared_error: 0.0661 - val_mean_absolute_error:
0.2160
Epoch 122/500
12/12 [==============================] - 0s 8ms/step - loss: 0.1122 -
mean_squared_error: 0.1122 - mean_absolute_error: 0.2358 - val_loss:
0.0654 - val_mean_squared_error: 0.0654 - val_mean_absolute_error:
0.2196
Epoch 123/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1119 -
mean_squared_error: 0.1119 - mean_absolute_error: 0.2375 - val_loss:
0.0657 - val_mean_squared_error: 0.0657 - val_mean_absolute_error:
0.2189
Epoch 124/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1143 -
mean_squared_error: 0.1143 - mean_absolute_error: 0.2369 - val_loss:
0.0663 - val_mean_squared_error: 0.0663 - val_mean_absolute_error:
0.2205
Epoch 125/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1021 -
mean_squared_error: 0.1021 - mean_absolute_error: 0.2246 - val_loss:
0.0654 - val_mean_squared_error: 0.0654 - val_mean_absolute_error:
0.2169
Epoch 126/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1265 -
mean_squared_error: 0.1265 - mean_absolute_error: 0.2453 - val_loss:
0.0681 - val_mean_squared_error: 0.0681 - val_mean_absolute_error:
0.2263
Epoch 127/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1115 -
mean_squared_error: 0.1115 - mean_absolute_error: 0.2329 - val_loss:
```

```
0.0633 - val_mean_squared_error: 0.0633 - val_mean_absolute_error:
0.2123
Epoch 128/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1155 -
mean_squared_error: 0.1155 - mean_absolute_error: 0.2420 - val_loss:
0.0657 - val_mean_squared_error: 0.0657 - val_mean_absolute_error:
0.2203
Epoch 129/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1060 -
mean_squared_error: 0.1060 - mean_absolute_error: 0.2256 - val_loss:
0.0662 - val_mean_squared_error: 0.0662 - val_mean_absolute_error:
0.2210
Epoch 130/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1050 -
mean_squared_error: 0.1050 - mean_absolute_error: 0.2265 - val_loss:
0.0662 - val_mean_squared_error: 0.0662 - val_mean_absolute_error:
0.2194
Epoch 131/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1031 -
mean_squared_error: 0.1031 - mean_absolute_error: 0.2275 - val_loss:
0.0643 - val_mean_squared_error: 0.0643 - val_mean_absolute_error:
0.2200
Epoch 132/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1007 -
mean_squared_error: 0.1007 - mean_absolute_error: 0.2219 - val_loss:
0.0639 - val_mean_squared_error: 0.0639 - val_mean_absolute_error:
0.2179
Epoch 133/500
12/12 [==============================] - 0s 7ms/step - loss: 0.0963 -
mean_squared_error: 0.0963 - mean_absolute_error: 0.2138 - val_loss:
0.0650 - val_mean_squared_error: 0.0650 - val_mean_absolute_error:
0.2189
Epoch 134/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1027 -
mean_squared_error: 0.1027 - mean_absolute_error: 0.2249 - val_loss:
0.0659 - val_mean_squared_error: 0.0659 - val_mean_absolute_error:
0.2193
Epoch 135/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1091 -
mean_squared_error: 0.1091 - mean_absolute_error: 0.2298 - val_loss:
0.0672 - val_mean_squared_error: 0.0672 - val_mean_absolute_error:
0.2227
Epoch 136/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1132 -
mean_squared_error: 0.1132 - mean_absolute_error: 0.2274 - val_loss:
0.0676 - val_mean_squared_error: 0.0676 - val_mean_absolute_error:
0.2238
Epoch 137/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1084 -
```

```
mean_squared_error: 0.1084 - mean_absolute_error: 0.2299 - val_loss:
0.0645 - val_mean_squared_error: 0.0645 - val_mean_absolute_error:
0.2183
Epoch 138/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1083 -
mean_squared_error: 0.1083 - mean_absolute_error: 0.2328 - val_loss:
0.0670 - val_mean_squared_error: 0.0670 - val_mean_absolute_error:
0.2231
Epoch 139/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1062 -
mean_squared_error: 0.1062 - mean_absolute_error: 0.2290 - val_loss:
0.0684 - val_mean_squared_error: 0.0684 - val_mean_absolute_error:
0.2266
Epoch 140/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1225 -
mean_squared_error: 0.1225 - mean_absolute_error: 0.2294 - val_loss:
0.0639 - val_mean_squared_error: 0.0639 - val_mean_absolute_error:
0.2163
Epoch 141/500
12/12 [==============================] - 0s 6ms/step - loss: 0.0943 -
mean_squared_error: 0.0943 - mean_absolute_error: 0.2148 - val_loss:
0.0656 - val_mean_squared_error: 0.0656 - val_mean_absolute_error:
0.2244
Epoch 142/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1116 -
mean_squared_error: 0.1116 - mean_absolute_error: 0.2322 - val_loss:
0.0626 - val_mean_squared_error: 0.0626 - val_mean_absolute_error:
0.2176
Epoch 143/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1031 -
mean_squared_error: 0.1031 - mean_absolute_error: 0.2277 - val_loss:
0.0635 - val_mean_squared_error: 0.0635 - val_mean_absolute_error:
0.2169
Epoch 144/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1050 -
mean_squared_error: 0.1050 - mean_absolute_error: 0.2288 - val_loss:
0.0653 - val_mean_squared_error: 0.0653 - val_mean_absolute_error:
0.2207
Epoch 145/500
12/12 [==============================] - 0s 8ms/step - loss: 0.0995 -
mean_squared_error: 0.0995 - mean_absolute_error: 0.2225 - val_loss:
0.0660 - val_mean_squared_error: 0.0660 - val_mean_absolute_error:
0.2217
Epoch 146/500
12/12 [==============================] - 0s 7ms/step - loss: 0.0997 -
mean_squared_error: 0.0997 - mean_absolute_error: 0.2223 - val_loss:
0.0643 - val_mean_squared_error: 0.0643 - val_mean_absolute_error:
0.2185
Epoch 147/500
```

```
12/12 [==============================] - 0s 5ms/step - loss: 0.0970 -
mean_squared_error: 0.0970 - mean_absolute_error: 0.2239 - val_loss:
0.0646 - val_mean_squared_error: 0.0646 - val_mean_absolute_error:
0.2222
Epoch 148/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1004 -
mean_squared_error: 0.1004 - mean_absolute_error: 0.2279 - val_loss:
0.0646 - val_mean_squared_error: 0.0646 - val_mean_absolute_error:
0.2220
Epoch 149/500
12/12 [==============================] - 0s 6ms/step - loss: 0.0964 -
mean_squared_error: 0.0964 - mean_absolute_error: 0.2129 - val_loss:
0.0649 - val_mean_squared_error: 0.0649 - val_mean_absolute_error:
0.2232
Epoch 150/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1007 -
mean_squared_error: 0.1007 - mean_absolute_error: 0.2129 - val_loss:
0.0637 - val_mean_squared_error: 0.0637 - val_mean_absolute_error:
0.2217
Epoch 151/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1053 -
mean_squared_error: 0.1053 - mean_absolute_error: 0.2248 - val_loss:
0.0640 - val_mean_squared_error: 0.0640 - val_mean_absolute_error:
0.2200
Epoch 152/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1050 -
mean_squared_error: 0.1050 - mean_absolute_error: 0.2218 - val_loss:
0.0665 - val_mean_squared_error: 0.0665 - val_mean_absolute_error:
0.2243
Epoch 153/500
12/12 [==============================] - 0s 7ms/step - loss: 0.0953 -
mean_squared_error: 0.0953 - mean_absolute_error: 0.2194 - val_loss:
0.0658 - val_mean_squared_error: 0.0658 - val_mean_absolute_error:
0.2196
Epoch 154/500
12/12 [==============================] - 0s 6ms/step - loss: 0.1013 -
mean_squared_error: 0.1013 - mean_absolute_error: 0.2335 - val_loss:
0.0658 - val_mean_squared_error: 0.0658 - val_mean_absolute_error:
0.2188
Epoch 155/500
12/12 [==============================] - 0s 5ms/step - loss: 0.0923 -
mean_squared_error: 0.0923 - mean_absolute_error: 0.2074 - val_loss:
0.0662 - val_mean_squared_error: 0.0662 - val_mean_absolute_error:
0.2199
Epoch 156/500
12/12 [==============================] - 0s 10ms/step - loss: 0.0897 -
mean_squared_error: 0.0897 - mean_absolute_error: 0.2165 - val_loss:
0.0679 - val_mean_squared_error: 0.0679 - val_mean_absolute_error:
0.2260
```

```
Epoch 157/500
12/12 [==============================] - 0s 9ms/step - loss: 0.0986 -
mean_squared_error: 0.0986 - mean_absolute_error: 0.2144 - val_loss:
0.0634 - val_mean_squared_error: 0.0634 - val_mean_absolute_error:
0.2172
Epoch 158/500
12/12 [==============================] - 0s 8ms/step - loss: 0.0955 -
mean_squared_error: 0.0955 - mean_absolute_error: 0.2245 - val_loss:
0.0632 - val_mean_squared_error: 0.0632 - val_mean_absolute_error:
0.2184
Epoch 159/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1046 -
mean_squared_error: 0.1046 - mean_absolute_error: 0.2313 - val_loss:
0.0684 - val_mean_squared_error: 0.0684 - val_mean_absolute_error:
0.2286
Epoch 160/500
12/12 [==============================] - 0s 9ms/step - loss: 0.1122 -
mean_squared_error: 0.1122 - mean_absolute_error: 0.2321 - val_loss:
0.0657 - val_mean_squared_error: 0.0657 - val_mean_absolute_error:
0.2227
Epoch 161/500
12/12 [==============================] - 0s 9ms/step - loss: 0.1053 -
mean_squared_error: 0.1053 - mean_absolute_error: 0.2205 - val_loss:
0.0643 - val_mean_squared_error: 0.0643 - val_mean_absolute_error:
0.2189
Epoch 162/500
12/12 [==============================] - 0s 7ms/step - loss: 0.1058 -
mean_squared_error: 0.1058 - mean_absolute_error: 0.2274 - val_loss:
0.0638 - val_mean_squared_error: 0.0638 - val_mean_absolute_error:
0.2163
```



Titanic Model

load data and check nu of nas and na percentages

```
titanic_train_data = pd.read_csv('/content/drive/My
Drive/InterpretableAI/Datasets/titanic/train.csv')
titanic_test_data = pd.read_csv('/content/drive/My
Drive/InterpretableAI/Datasets/titanic/test.csv')

print(titanic_train_data.shape)
print(titanic_test_data.shape)

# Check NA values in training data
print("NA values in training data:")
print(titanic_train_data.isna().sum())

print("\nNA values in test data:")
print(titanic_test_data.isna().sum())

# Percentage of NA values in training data
print("\nPercentage of NA values in training data:")
print((titanic_train_data.isna().sum() / len(titanic_train_data)) *
100)
```

```
print("\nPercentage of NA values in test data:")
print((titanic_test_data.isna().sum() / len(titanic_test_data)) * 100)

print(titanic_train_data.head())
print(titanic_test_data.head())
```

```
(891, 12)
(418, 11)
NA values in training data:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

NA values in test data:
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64

Percentage of NA values in training data:
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
```

```
dtype: float64

Percentage of NA values in test data:
PassengerId     0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            20.574163
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.239234
Cabin          78.229665
Embarked        0.000000
dtype: float64
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
   PassengerId  Pclass                                          Name
Sex  \
0          892       3                              Kelly, Mr. James
male
1          893       3              Wilkes, Mrs. James (Ellen Needs)
female
2          894       2                     Myles, Mr. Thomas Francis
male
3          895       3                              Wirz, Mr. Albert
```

```
male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)
female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

Drop non-numerical columns

```
# Feature engineering

# Drop unnecessary columns
titanic_train_data = titanic_train_data.drop(columns = ['Name',
'Ticket', 'Cabin', 'PassengerId'])
titanic_test_data = titanic_test_data.drop(columns = ['Name',
'Ticket', 'Cabin', 'PassengerId'])

print(titanic_train_data.head())
print(titanic_test_data.head())
```

```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
0         0       3    male  22.0      1      0   7.2500        S
1         1       1  female  38.0      1      0  71.2833        C
2         1       3  female  26.0      0      0   7.9250        S
3         1       1  female  35.0      1      0  53.1000        S
4         0       3    male  35.0      0      0   8.0500        S
   Pclass     Sex   Age  SibSp  Parch     Fare Embarked
0       3    male  34.5      0      0   7.8292        Q
1       3  female  47.0      1      0   7.0000        S
2       2    male  62.0      0      0   9.6875        Q
3       3    male  27.0      0      0   8.6625        S
4       3  female  22.0      1      1  12.2875        S
```

Encode categorical columns

```
def encode_features(data):
  mappings = {}
  for column in data.columns:
    if data[column].dtype == 'object':
      le = LabelEncoder()
      data[column] = le.fit_transform(data[column])
      # Store the mapping
      mappings[column] = dict(zip(le.classes_,
le.transform(le.classes_)))
      print(f"\nMapping for {column}:")
```

```
      for original, encoded in mappings[column].items():
        print(f"{original} → {encoded}")
  return data

titanic_train_data = encode_features(titanic_train_data)
titanic_test_data = encode_features(titanic_test_data)


print(titanic_train_data.head())
print(titanic_test_data.head())


Mapping for Sex:
female → 0
male → 1

Mapping for Embarked:
C → 0
Q → 1
S → 2
nan → 3

Mapping for Sex:
female → 0
male → 1

Mapping for Embarked:
C → 0
Q → 1
S → 2
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
0         0       3    1  22.0      1      0   7.2500         2
1         1       1    0  38.0      1      0  71.2833         0
2         1       3    0  26.0      0      0   7.9250         2
3         1       1    0  35.0      1      0  53.1000         2
4         0       3    1  35.0      0      0   8.0500         2
   Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
0       3    1  34.5      0      0   7.8292         1
1       3    0  47.0      1      0   7.0000         2
2       2    1  62.0      0      0   9.6875         1
3       3    1  27.0      0      0   8.6625         2
4       3    0  22.0      1      1  12.2875         2
```

Deal with NA values by replacing them with their means and dropping the 2 in embarked col

```
titanic_train_data = titanic_train_data[titanic_train_data['Embarked']
!= 3]

titanic_train_data =
```

```
titanic_train_data.fillna(titanic_train_data.mean())
titanic_test_data = titanic_test_data.fillna(titanic_test_data.mean())
```

Scale non-categorical values using z-score

```
titanic_train_data = z_score_scale(titanic_train_data, ['Age', 'Fare',
'SibSp', 'Parch'])
titanic_test_data = z_score_scale(titanic_test_data, ['Age', 'Fare',
'SibSp', 'Parch'])

print(titanic_train_data.head())
print(titanic_test_data.head())
```

```
   Survived  Pclass  Sex       Age     SibSp     Parch      Fare
Embarked
0         0       3    1 -0.589620  0.431350 -0.474326 -0.500240
2
1         1       1    0  0.644848  0.431350 -0.474326  0.788947
0
2         1       3    0 -0.281003 -0.475199 -0.474326 -0.486650
2
3         1       1    0  0.413385  0.431350 -0.474326  0.422861
2
4         0       3    1  0.413385 -0.475199 -0.474326 -0.484133
2
   Pclass  Sex       Age     SibSp     Parch      Fare  Embarked
0       3    1  0.334993 -0.499470 -0.400248 -0.498407         1
1       3    0  1.325530  0.616992 -0.400248 -0.513274         2
2       2    1  2.514175 -0.499470 -0.400248 -0.465088         1
3       3    1 -0.259330 -0.499470 -0.400248 -0.483466         2
4       3    0 -0.655545  0.616992  0.619896 -0.418471         2
```

Separate labels and features for training

```
titanic_train_features = titanic_train_data.drop(columns =
['Survived'])
titanic_train_labels = titanic_train_data['Survived'].values

titanic_test_features = titanic_test_data
```

Train the Model

```
# One-hot encode the reset labels
titanic_train_labels = to_categorical(titanic_train_labels,
num_classes=2)
print(f"One-hot encoded labels shape: {titanic_train_labels.shape}")

One-hot encoded labels shape: (889, 2)
```

```
print(f"Features shape: {titanic_train_features.shape}")
print(f"Labels shape: {titanic_train_labels.shape}")
```

```
Features shape: (889, 7)
Labels shape: (889, 2)
```

```
titanic_model = tf.keras.Sequential([
  tf.keras.layers.Dense(32, activation = 'relu'),
  tf.keras.layers.Dropout(0.1),
  tf.keras.layers.Dense(2, activation = 'softmax')
])

titanic_model.compile(optimizer = 'adam',
           loss = 'categorical_crossentropy',
           metrics = ['acc'])

history = titanic_model.fit(
    titanic_train_features.values,
    titanic_train_labels,
    epochs=500,
    validation_split=0.1,
    batch_size=32,
    callbacks=[early_stop]
)


plotter = tfdocs.plots.HistoryPlotter(metric = 'acc')
plotter.plot({'Basic': history}, metric = 'acc')
plt.ylim([0, 1])
plt.ylabel('Accuracy')
plt.show()
```

```
Epoch 1/500
25/25 [==============================] - 2s 21ms/step - loss: 0.7812 -
acc: 0.5487 - val_loss: 0.6628 - val_acc: 0.6292
Epoch 2/500
25/25 [==============================] - 0s 6ms/step - loss: 0.7196 -
acc: 0.6075 - val_loss: 0.6301 - val_acc: 0.6292
Epoch 3/500
25/25 [==============================] - 0s 5ms/step - loss: 0.6841 -
acc: 0.6162 - val_loss: 0.6071 - val_acc: 0.6180
Epoch 4/500
25/25 [==============================] - 0s 5ms/step - loss: 0.6484 -
acc: 0.6212 - val_loss: 0.5842 - val_acc: 0.6180
Epoch 5/500
25/25 [==============================] - 0s 7ms/step - loss: 0.5986 -
acc: 0.6525 - val_loss: 0.5593 - val_acc: 0.6629
Epoch 6/500
25/25 [==============================] - 0s 7ms/step - loss: 0.5824 -
acc: 0.6975 - val_loss: 0.5397 - val_acc: 0.7079
```

```
Epoch 7/500
25/25 [==============================] - 0s 8ms/step - loss: 0.5610 -
acc: 0.7212 - val_loss: 0.5163 - val_acc: 0.7640
Epoch 8/500
25/25 [==============================] - 0s 7ms/step - loss: 0.5356 -
acc: 0.7475 - val_loss: 0.5012 - val_acc: 0.7865
Epoch 9/500
25/25 [==============================] - 0s 7ms/step - loss: 0.5262 -
acc: 0.7487 - val_loss: 0.4878 - val_acc: 0.8090
Epoch 10/500
25/25 [==============================] - 0s 4ms/step - loss: 0.5043 -
acc: 0.7613 - val_loss: 0.4739 - val_acc: 0.7865
Epoch 11/500
25/25 [==============================] - 0s 4ms/step - loss: 0.5010 -
acc: 0.7763 - val_loss: 0.4670 - val_acc: 0.8315
Epoch 12/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4907 -
acc: 0.7738 - val_loss: 0.4565 - val_acc: 0.8315
Epoch 13/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4760 -
acc: 0.7950 - val_loss: 0.4498 - val_acc: 0.8315
Epoch 14/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4703 -
acc: 0.8025 - val_loss: 0.4441 - val_acc: 0.8315
Epoch 15/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4672 -
acc: 0.7962 - val_loss: 0.4388 - val_acc: 0.8315
Epoch 16/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4773 -
acc: 0.7812 - val_loss: 0.4355 - val_acc: 0.8427
Epoch 17/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4716 -
acc: 0.7862 - val_loss: 0.4326 - val_acc: 0.8427
Epoch 18/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4726 -
acc: 0.7900 - val_loss: 0.4285 - val_acc: 0.8427
Epoch 19/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4644 -
acc: 0.8037 - val_loss: 0.4262 - val_acc: 0.8539
Epoch 20/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4612 -
acc: 0.8012 - val_loss: 0.4255 - val_acc: 0.8539
Epoch 21/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4682 -
acc: 0.8000 - val_loss: 0.4219 - val_acc: 0.8539
Epoch 22/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4711 -
acc: 0.7950 - val_loss: 0.4209 - val_acc: 0.8539
Epoch 23/500
```

```
25/25 [==============================] - 0s 4ms/step - loss: 0.4647 -
acc: 0.7975 - val_loss: 0.4167 - val_acc: 0.8539
Epoch 24/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4555 -
acc: 0.8012 - val_loss: 0.4153 - val_acc: 0.8539
Epoch 25/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4494 -
acc: 0.8112 - val_loss: 0.4132 - val_acc: 0.8539
Epoch 26/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4531 -
acc: 0.8138 - val_loss: 0.4134 - val_acc: 0.8539
Epoch 27/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4558 -
acc: 0.7950 - val_loss: 0.4123 - val_acc: 0.8539
Epoch 28/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4493 -
acc: 0.8037 - val_loss: 0.4094 - val_acc: 0.8539
Epoch 29/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4470 -
acc: 0.7975 - val_loss: 0.4098 - val_acc: 0.8539
Epoch 30/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4388 -
acc: 0.8100 - val_loss: 0.4072 - val_acc: 0.8539
Epoch 31/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4411 -
acc: 0.8112 - val_loss: 0.4075 - val_acc: 0.8539
Epoch 32/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4492 -
acc: 0.8150 - val_loss: 0.4067 - val_acc: 0.8539
Epoch 33/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4424 -
acc: 0.8050 - val_loss: 0.4020 - val_acc: 0.8539
Epoch 34/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4542 -
acc: 0.8075 - val_loss: 0.4003 - val_acc: 0.8539
Epoch 35/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4426 -
acc: 0.8112 - val_loss: 0.4026 - val_acc: 0.8539
Epoch 36/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4415 -
acc: 0.8037 - val_loss: 0.4031 - val_acc: 0.8539
Epoch 37/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4487 -
acc: 0.8075 - val_loss: 0.3986 - val_acc: 0.8539
Epoch 38/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4430 -
acc: 0.8100 - val_loss: 0.3977 - val_acc: 0.8539
Epoch 39/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4444 -
```

```
acc: 0.8037 - val_loss: 0.3985 - val_acc: 0.8539
Epoch 40/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4422 -
acc: 0.8037 - val_loss: 0.3977 - val_acc: 0.8539
Epoch 41/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4350 -
acc: 0.8150 - val_loss: 0.3949 - val_acc: 0.8539
Epoch 42/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4285 -
acc: 0.8163 - val_loss: 0.3934 - val_acc: 0.8539
Epoch 43/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4410 -
acc: 0.8150 - val_loss: 0.3921 - val_acc: 0.8539
Epoch 44/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4266 -
acc: 0.8175 - val_loss: 0.3931 - val_acc: 0.8652
Epoch 45/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4376 -
acc: 0.8112 - val_loss: 0.3932 - val_acc: 0.8539
Epoch 46/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4347 -
acc: 0.8075 - val_loss: 0.3944 - val_acc: 0.8652
Epoch 47/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4312 -
acc: 0.8150 - val_loss: 0.3911 - val_acc: 0.8539
Epoch 48/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4282 -
acc: 0.8125 - val_loss: 0.3912 - val_acc: 0.8652
Epoch 49/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4265 -
acc: 0.8300 - val_loss: 0.3899 - val_acc: 0.8539
Epoch 50/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4354 -
acc: 0.8138 - val_loss: 0.3902 - val_acc: 0.8539
Epoch 51/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4224 -
acc: 0.8175 - val_loss: 0.3912 - val_acc: 0.8539
Epoch 52/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4313 -
acc: 0.8112 - val_loss: 0.3887 - val_acc: 0.8652
Epoch 53/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4244 -
acc: 0.8213 - val_loss: 0.3884 - val_acc: 0.8652
Epoch 54/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4218 -
acc: 0.8150 - val_loss: 0.3868 - val_acc: 0.8539
Epoch 55/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4268 -
acc: 0.8175 - val_loss: 0.3870 - val_acc: 0.8652
```

```
Epoch 56/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4340 -
acc: 0.8188 - val_loss: 0.3864 - val_acc: 0.8652
Epoch 57/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4300 -
acc: 0.8163 - val_loss: 0.3867 - val_acc: 0.8652
Epoch 58/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4280 -
acc: 0.8200 - val_loss: 0.3844 - val_acc: 0.8652
Epoch 59/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4352 -
acc: 0.8225 - val_loss: 0.3857 - val_acc: 0.8652
Epoch 60/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4287 -
acc: 0.8200 - val_loss: 0.3856 - val_acc: 0.8652
Epoch 61/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4272 -
acc: 0.8275 - val_loss: 0.3838 - val_acc: 0.8652
Epoch 62/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4252 -
acc: 0.8163 - val_loss: 0.3868 - val_acc: 0.8652
Epoch 63/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4406 -
acc: 0.8125 - val_loss: 0.3832 - val_acc: 0.8652
Epoch 64/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4210 -
acc: 0.8275 - val_loss: 0.3866 - val_acc: 0.8652
Epoch 65/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4239 -
acc: 0.8263 - val_loss: 0.3841 - val_acc: 0.8652
Epoch 66/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4274 -
acc: 0.8213 - val_loss: 0.3836 - val_acc: 0.8652
Epoch 67/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4233 -
acc: 0.8325 - val_loss: 0.3816 - val_acc: 0.8652
Epoch 68/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4305 -
acc: 0.8163 - val_loss: 0.3837 - val_acc: 0.8652
Epoch 69/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4158 -
acc: 0.8225 - val_loss: 0.3827 - val_acc: 0.8652
Epoch 70/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4244 -
acc: 0.8163 - val_loss: 0.3833 - val_acc: 0.8652
Epoch 71/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4232 -
acc: 0.8238 - val_loss: 0.3801 - val_acc: 0.8652
Epoch 72/500
```

```
25/25 [==============================] - 0s 4ms/step - loss: 0.4213 -
acc: 0.8250 - val_loss: 0.3805 - val_acc: 0.8652
Epoch 73/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4204 -
acc: 0.8100 - val_loss: 0.3804 - val_acc: 0.8652
Epoch 74/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4219 -
acc: 0.8225 - val_loss: 0.3806 - val_acc: 0.8652
Epoch 75/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4216 -
acc: 0.8200 - val_loss: 0.3815 - val_acc: 0.8652
Epoch 76/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4233 -
acc: 0.8150 - val_loss: 0.3796 - val_acc: 0.8652
Epoch 77/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4229 -
acc: 0.8238 - val_loss: 0.3807 - val_acc: 0.8652
Epoch 78/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4202 -
acc: 0.8275 - val_loss: 0.3783 - val_acc: 0.8652
Epoch 79/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4259 -
acc: 0.8112 - val_loss: 0.3772 - val_acc: 0.8652
Epoch 80/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4162 -
acc: 0.8275 - val_loss: 0.3756 - val_acc: 0.8652
Epoch 81/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4161 -
acc: 0.8275 - val_loss: 0.3765 - val_acc: 0.8652
Epoch 82/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4092 -
acc: 0.8250 - val_loss: 0.3765 - val_acc: 0.8652
Epoch 83/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4067 -
acc: 0.8250 - val_loss: 0.3768 - val_acc: 0.8652
Epoch 84/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4120 -
acc: 0.8112 - val_loss: 0.3767 - val_acc: 0.8652
Epoch 85/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4165 -
acc: 0.8275 - val_loss: 0.3747 - val_acc: 0.8652
Epoch 86/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4168 -
acc: 0.8275 - val_loss: 0.3755 - val_acc: 0.8652
Epoch 87/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4184 -
acc: 0.8175 - val_loss: 0.3788 - val_acc: 0.8652
Epoch 88/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4128 -
```

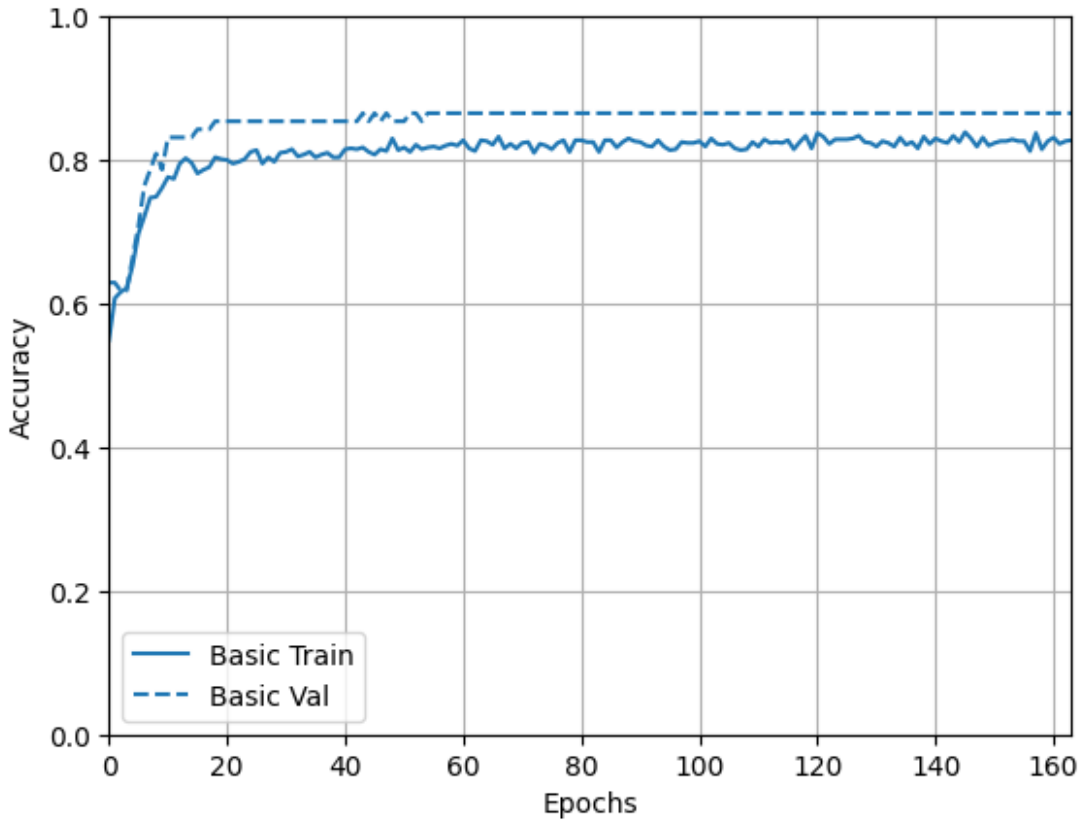
```
acc: 0.8250 - val_loss: 0.3751 - val_acc: 0.8652
Epoch 89/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4129 -
acc: 0.8300 - val_loss: 0.3767 - val_acc: 0.8652
Epoch 90/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4065 -
acc: 0.8263 - val_loss: 0.3752 - val_acc: 0.8652
Epoch 91/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4210 -
acc: 0.8250 - val_loss: 0.3753 - val_acc: 0.8652
Epoch 92/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4177 -
acc: 0.8200 - val_loss: 0.3755 - val_acc: 0.8652
Epoch 93/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4175 -
acc: 0.8188 - val_loss: 0.3756 - val_acc: 0.8652
Epoch 94/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4126 -
acc: 0.8275 - val_loss: 0.3750 - val_acc: 0.8652
Epoch 95/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4128 -
acc: 0.8188 - val_loss: 0.3764 - val_acc: 0.8652
Epoch 96/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4128 -
acc: 0.8138 - val_loss: 0.3740 - val_acc: 0.8652
Epoch 97/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4180 -
acc: 0.8150 - val_loss: 0.3738 - val_acc: 0.8652
Epoch 98/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4126 -
acc: 0.8250 - val_loss: 0.3773 - val_acc: 0.8652
Epoch 99/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4256 -
acc: 0.8238 - val_loss: 0.3759 - val_acc: 0.8652
Epoch 100/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4173 -
acc: 0.8238 - val_loss: 0.3737 - val_acc: 0.8652
Epoch 101/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4120 -
acc: 0.8263 - val_loss: 0.3744 - val_acc: 0.8652
Epoch 102/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4179 -
acc: 0.8188 - val_loss: 0.3746 - val_acc: 0.8652
Epoch 103/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4127 -
acc: 0.8300 - val_loss: 0.3718 - val_acc: 0.8652
Epoch 104/500
25/25 [==============================] - 0s 7ms/step - loss: 0.4152 -
acc: 0.8225 - val_loss: 0.3742 - val_acc: 0.8652
```

```
Epoch 105/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4170 -
acc: 0.8213 - val_loss: 0.3727 - val_acc: 0.8652
Epoch 106/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4076 -
acc: 0.8238 - val_loss: 0.3728 - val_acc: 0.8652
Epoch 107/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4094 -
acc: 0.8163 - val_loss: 0.3730 - val_acc: 0.8652
Epoch 108/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4124 -
acc: 0.8138 - val_loss: 0.3748 - val_acc: 0.8652
Epoch 109/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4140 -
acc: 0.8150 - val_loss: 0.3719 - val_acc: 0.8652
Epoch 110/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4128 -
acc: 0.8250 - val_loss: 0.3737 - val_acc: 0.8652
Epoch 111/500
25/25 [==============================] - 0s 7ms/step - loss: 0.4136 -
acc: 0.8188 - val_loss: 0.3768 - val_acc: 0.8652
Epoch 112/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4003 -
acc: 0.8288 - val_loss: 0.3728 - val_acc: 0.8652
Epoch 113/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4093 -
acc: 0.8238 - val_loss: 0.3719 - val_acc: 0.8652
Epoch 114/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4094 -
acc: 0.8250 - val_loss: 0.3744 - val_acc: 0.8652
Epoch 115/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4057 -
acc: 0.8238 - val_loss: 0.3731 - val_acc: 0.8652
Epoch 116/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4052 -
acc: 0.8275 - val_loss: 0.3714 - val_acc: 0.8652
Epoch 117/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4154 -
acc: 0.8150 - val_loss: 0.3746 - val_acc: 0.8652
Epoch 118/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4102 -
acc: 0.8250 - val_loss: 0.3733 - val_acc: 0.8652
Epoch 119/500
25/25 [==============================] - 0s 7ms/step - loss: 0.4042 -
acc: 0.8338 - val_loss: 0.3740 - val_acc: 0.8652
Epoch 120/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4065 -
acc: 0.8163 - val_loss: 0.3739 - val_acc: 0.8652
Epoch 121/500
```

```
25/25 [==============================] - 0s 5ms/step - loss: 0.4027 -
acc: 0.8375 - val_loss: 0.3741 - val_acc: 0.8652
Epoch 122/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4063 -
acc: 0.8325 - val_loss: 0.3716 - val_acc: 0.8652
Epoch 123/500
25/25 [==============================] - 0s 7ms/step - loss: 0.4106 -
acc: 0.8225 - val_loss: 0.3703 - val_acc: 0.8652
Epoch 124/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4139 -
acc: 0.8288 - val_loss: 0.3702 - val_acc: 0.8652
Epoch 125/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4067 -
acc: 0.8288 - val_loss: 0.3703 - val_acc: 0.8652
Epoch 126/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4061 -
acc: 0.8288 - val_loss: 0.3696 - val_acc: 0.8652
Epoch 127/500
25/25 [==============================] - 0s 5ms/step - loss: 0.3994 -
acc: 0.8300 - val_loss: 0.3713 - val_acc: 0.8652
Epoch 128/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4088 -
acc: 0.8338 - val_loss: 0.3716 - val_acc: 0.8652
Epoch 129/500
25/25 [==============================] - 0s 6ms/step - loss: 0.3965 -
acc: 0.8250 - val_loss: 0.3707 - val_acc: 0.8652
Epoch 130/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4039 -
acc: 0.8238 - val_loss: 0.3700 - val_acc: 0.8652
Epoch 131/500
25/25 [==============================] - 0s 7ms/step - loss: 0.4103 -
acc: 0.8188 - val_loss: 0.3671 - val_acc: 0.8652
Epoch 132/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4054 -
acc: 0.8263 - val_loss: 0.3676 - val_acc: 0.8652
Epoch 133/500
25/25 [==============================] - 0s 6ms/step - loss: 0.3990 -
acc: 0.8238 - val_loss: 0.3693 - val_acc: 0.8652
Epoch 134/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4079 -
acc: 0.8188 - val_loss: 0.3685 - val_acc: 0.8652
Epoch 135/500
25/25 [==============================] - 0s 6ms/step - loss: 0.4015 -
acc: 0.8313 - val_loss: 0.3674 - val_acc: 0.8652
Epoch 136/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4015 -
acc: 0.8213 - val_loss: 0.3676 - val_acc: 0.8652
Epoch 137/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4103 -
```

```
acc: 0.8250 - val_loss: 0.3664 - val_acc: 0.8652
Epoch 138/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4040 -
acc: 0.8163 - val_loss: 0.3652 - val_acc: 0.8652
Epoch 139/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4025 -
acc: 0.8325 - val_loss: 0.3671 - val_acc: 0.8652
Epoch 140/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4079 -
acc: 0.8225 - val_loss: 0.3671 - val_acc: 0.8652
Epoch 141/500
25/25 [==============================] - 0s 5ms/step - loss: 0.3968 -
acc: 0.8300 - val_loss: 0.3664 - val_acc: 0.8652
Epoch 142/500
25/25 [==============================] - 0s 3ms/step - loss: 0.4085 -
acc: 0.8263 - val_loss: 0.3681 - val_acc: 0.8652
Epoch 143/500
25/25 [==============================] - 0s 4ms/step - loss: 0.3978 -
acc: 0.8238 - val_loss: 0.3690 - val_acc: 0.8652
Epoch 144/500
25/25 [==============================] - 0s 3ms/step - loss: 0.3975 -
acc: 0.8338 - val_loss: 0.3651 - val_acc: 0.8652
Epoch 145/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4072 -
acc: 0.8250 - val_loss: 0.3690 - val_acc: 0.8652
Epoch 146/500
25/25 [==============================] - 0s 5ms/step - loss: 0.3995 -
acc: 0.8388 - val_loss: 0.3687 - val_acc: 0.8652
Epoch 147/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4065 -
acc: 0.8300 - val_loss: 0.3680 - val_acc: 0.8652
Epoch 148/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4064 -
acc: 0.8188 - val_loss: 0.3696 - val_acc: 0.8652
Epoch 149/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4008 -
acc: 0.8313 - val_loss: 0.3711 - val_acc: 0.8652
Epoch 150/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4077 -
acc: 0.8200 - val_loss: 0.3676 - val_acc: 0.8652
Epoch 151/500
25/25 [==============================] - 0s 4ms/step - loss: 0.3964 -
acc: 0.8238 - val_loss: 0.3666 - val_acc: 0.8652
Epoch 152/500
25/25 [==============================] - 0s 4ms/step - loss: 0.3977 -
acc: 0.8263 - val_loss: 0.3725 - val_acc: 0.8652
Epoch 153/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4027 -
acc: 0.8263 - val_loss: 0.3662 - val_acc: 0.8652
```

```
Epoch 154/500
25/25 [==============================] - 0s 5ms/step - loss: 0.3983 -
acc: 0.8288 - val_loss: 0.3723 - val_acc: 0.8652
Epoch 155/500
25/25 [==============================] - 0s 5ms/step - loss: 0.3962 -
acc: 0.8263 - val_loss: 0.3690 - val_acc: 0.8652
Epoch 156/500
25/25 [==============================] - 0s 3ms/step - loss: 0.3996 -
acc: 0.8263 - val_loss: 0.3714 - val_acc: 0.8652
Epoch 157/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4131 -
acc: 0.8125 - val_loss: 0.3710 - val_acc: 0.8652
Epoch 158/500
25/25 [==============================] - 0s 3ms/step - loss: 0.3937 -
acc: 0.8375 - val_loss: 0.3673 - val_acc: 0.8652
Epoch 159/500
25/25 [==============================] - 0s 4ms/step - loss: 0.4046 -
acc: 0.8150 - val_loss: 0.3684 - val_acc: 0.8652
Epoch 160/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4012 -
acc: 0.8263 - val_loss: 0.3695 - val_acc: 0.8652
Epoch 161/500
25/25 [==============================] - 0s 5ms/step - loss: 0.4023 -
acc: 0.8313 - val_loss: 0.3697 - val_acc: 0.8652
Epoch 162/500
25/25 [==============================] - 0s 3ms/step - loss: 0.3967 -
acc: 0.8225 - val_loss: 0.3720 - val_acc: 0.8652
Epoch 163/500
25/25 [==============================] - 0s 5ms/step - loss: 0.3992 -
acc: 0.8263 - val_loss: 0.3703 - val_acc: 0.8652
Epoch 164/500
25/25 [==============================] - 0s 5ms/step - loss: 0.3966 -
acc: 0.8275 - val_loss: 0.3701 - val_acc: 0.8652
```



Part 1 Feature Level Interpretability

1. Partial Dependence Plots (PDP) and Individual Conditional Expectation (ICE) plots

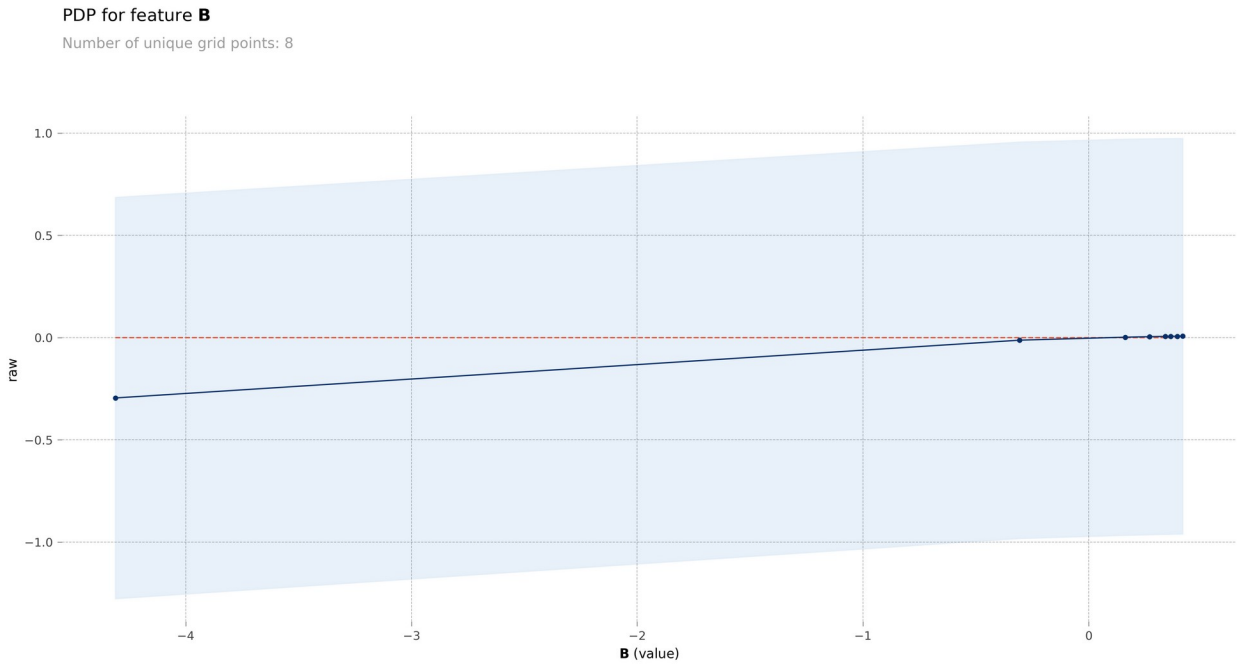
```
features_to_plot = ['B', 'LSTAT', 'NOX']

for i, feature in enumerate(features_to_plot):
    # Create the PDPBox isolate object
    pdp_isolate = pdp.PDPIsolate(
        model=boston_model,
        df=boston_test_df,
        model_features=boston_feature_names,
        feature=feature,
        feature_name=feature,
        n_classes=0
    )

    # Plot the PDP
    fig, axes = pdp_isolate.plot(
        center=False,
        plot_lines=False,
        plot_pts_dist=False,
        to_bins=False,
```

```
        engine='matplotlib'
    )
    plt.tight_layout()
    plt.show()



# PDPs for Titanic
features_to_plot = ['Age', 'Fare', 'Pclass']

for i, feature in enumerate(features_to_plot):
    # Create the PDPBox isolate object
    pdp_isolate = pdp.PDPIsolate(
        model=titanic_model,
        df=titanic_test_data,
        model_features=titanic_train_features.columns,
        feature=feature,
        feature_name=feature,
        n_classes=2
    )

    # Plot the PDP
    fig, axes = pdp_isolate.plot(
        center=False,
        plot_lines=False,
        plot_pts_dist=False,
        to_bins=False,
        engine='matplotlib'
    )

    plt.tight_layout()
    plt.show()
```

```
obtain pred_func from the provided model.
```

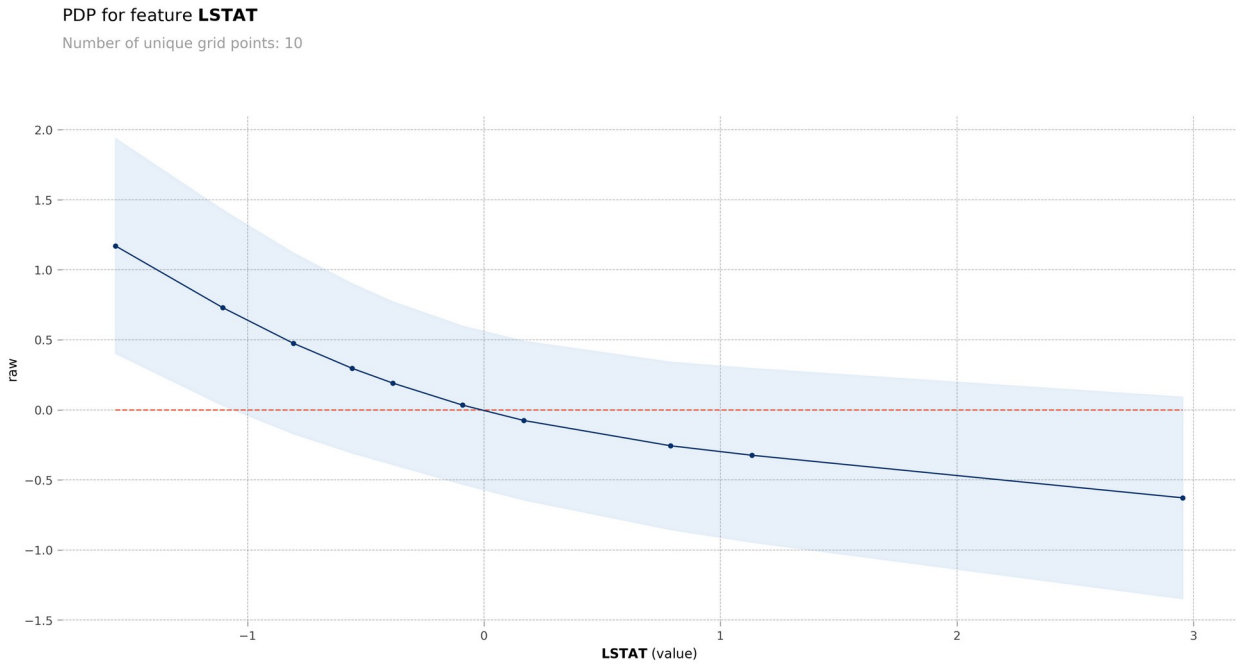

```
{"model_id":"cf6a02b3d5a44d1f9b6a6548cf52d0b5","version_major":2,"vers
ion_minor":0}
```


```
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family ['Arial'] not
found. Falling back to DejaVu Sans.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```

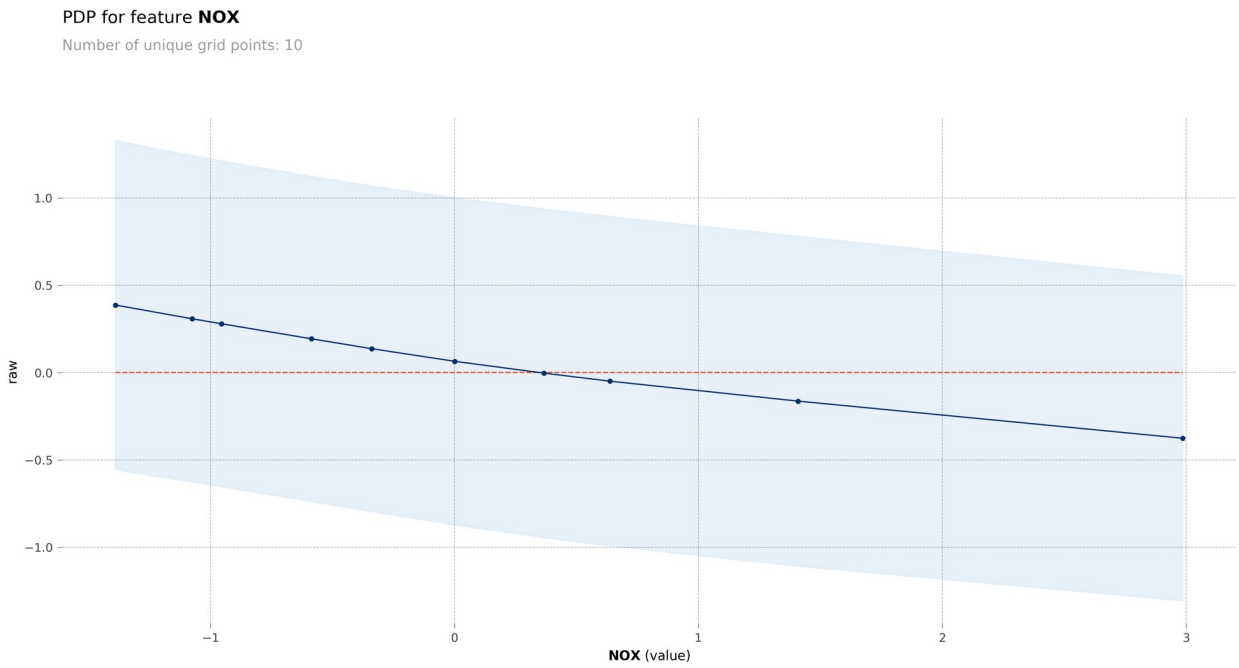


{"model_id":"3e144c8b4e5642c9bbfdd1af6309f621","version_major":2,"version_minor":0}

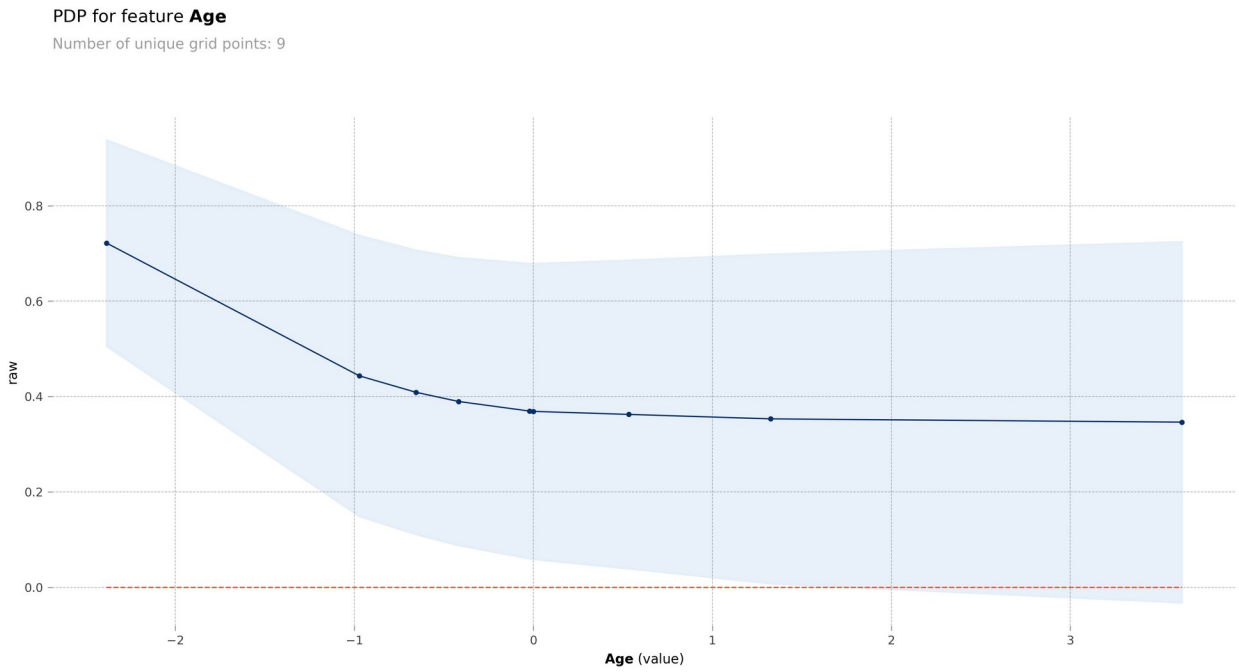
```
obtain pred_func from the provided model.

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
```



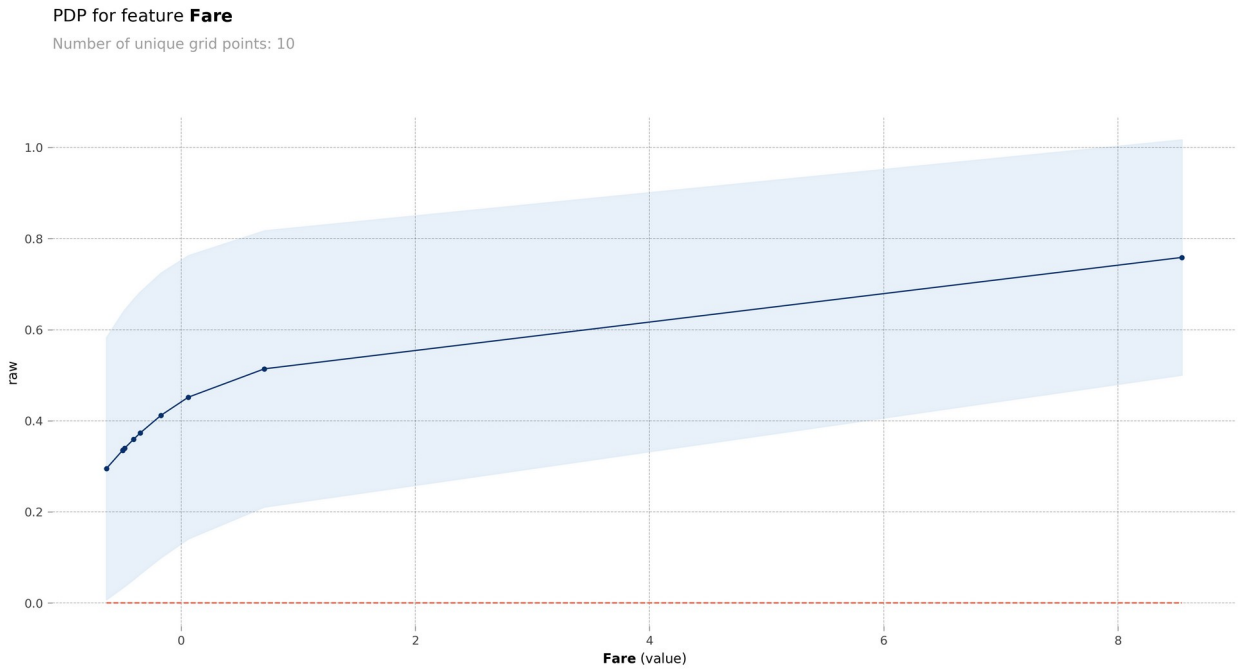

```
obtain pred_func from the provided model.

{"model_id":"cb895509cac84e13bb821b9c8392d74b","version_major":2,"vers
ion_minor":0}

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```



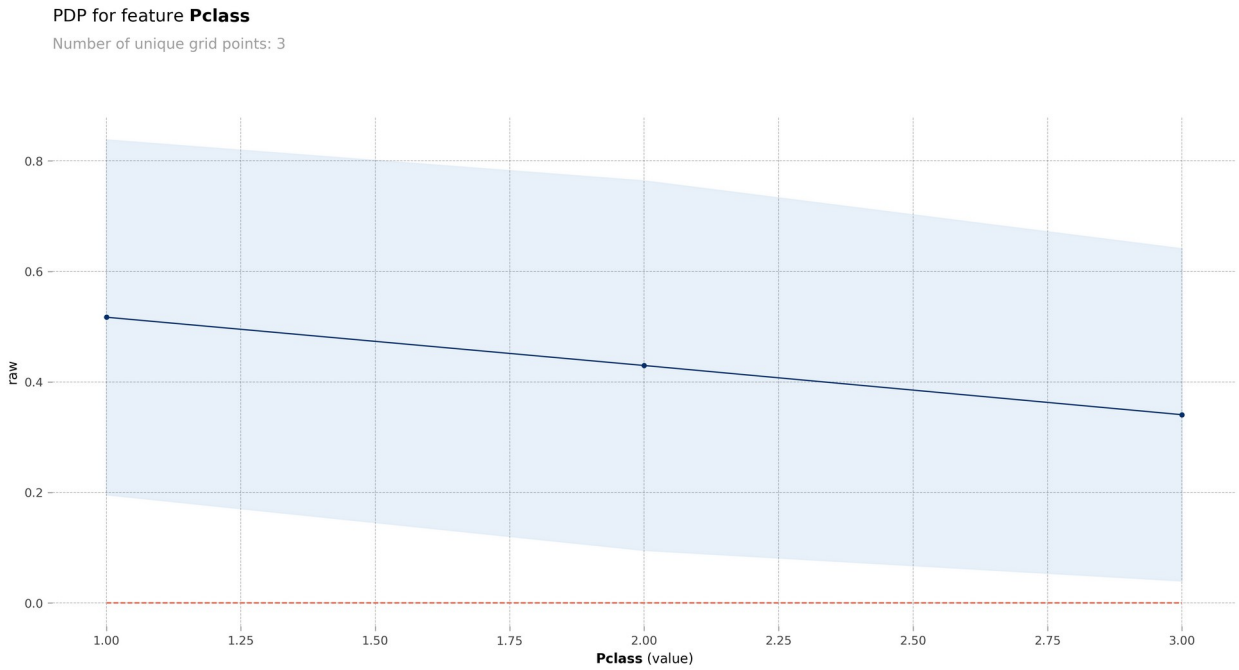

```
obtain pred_func from the provided model.

{"model_id":"a1e2bf6953bd45eca69bb4a7bac00176","version_major":2,"vers
ion_minor":0}

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```




```
obtain pred_func from the provided model.

{"model_id":"feffe1fee87f45ab95f16ddc76f24435","version_major":2,"vers
ion_minor":0}

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```




```
obtain pred_func from the provided model.

{"model_id":"c1c53268a41040b68b6d18c31981bf42","version_major":2,"vers
ion_minor":0}

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```



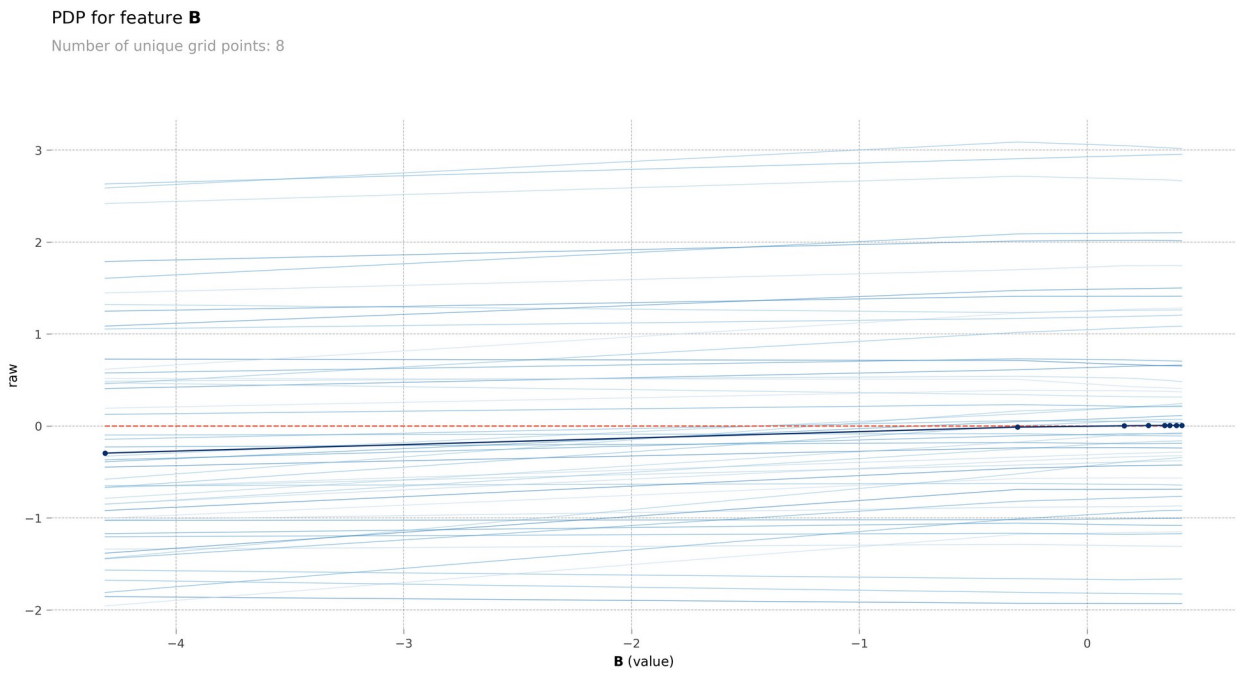

```
features_to_plot = ['B', 'LSTAT', 'NOX']

for i, feature in enumerate(features_to_plot):
    # Create the PDPBox isolate object
    pdp_isolate = pdp.PDPIsolate(
        model=boston_model,
        df=boston_test_df,
        model_features=boston_feature_names,
        feature=feature,
        feature_name=feature,
        n_classes=0
    )

    # Plot the ICE
    fig, axes = pdp_isolate.plot(
        center=False,
        plot_lines=True,
        plot_pts_dist=False,
        to_bins=False,
        cluster=True,
        engine='matplotlib',
        n_cluster_centers=50
    )
    plt.tight_layout()
    plt.show()


# ICE for Titanic
```

```
features_to_plot = ['Age', 'Fare', 'Pclass']

for i, feature in enumerate(features_to_plot):
    # Create the PDPBox isolate object
    pdp_isolate = pdp.PDPIsolate(
        model=titanic_model,
        df=titanic_test_data,
        model_features=titanic_train_features.columns,
        feature=feature,
        feature_name=feature,
        n_classes=2
    )

    # Plot the ICE
    fig, axes = pdp_isolate.plot(
        center=False,
        plot_lines=True,
        plot_pts_dist=False,
        to_bins=False,
        cluster=True,
        engine='matplotlib',
        n_cluster_centers=50
    )

    plt.tight_layout()
    plt.show()
```

```
obtain pred_func from the provided model.
```


```
{"model_id":"0b24369417444d8591a1d38643117684","version_major":2,"vers
ion_minor":0}
```

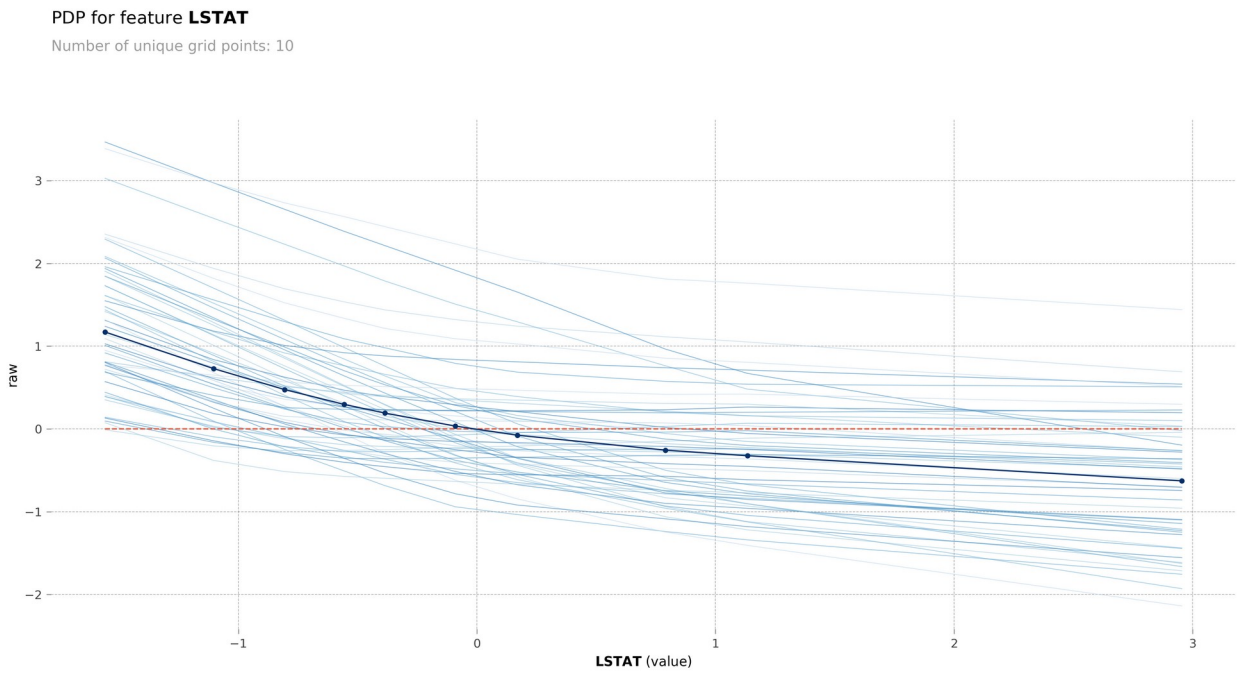

```
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```



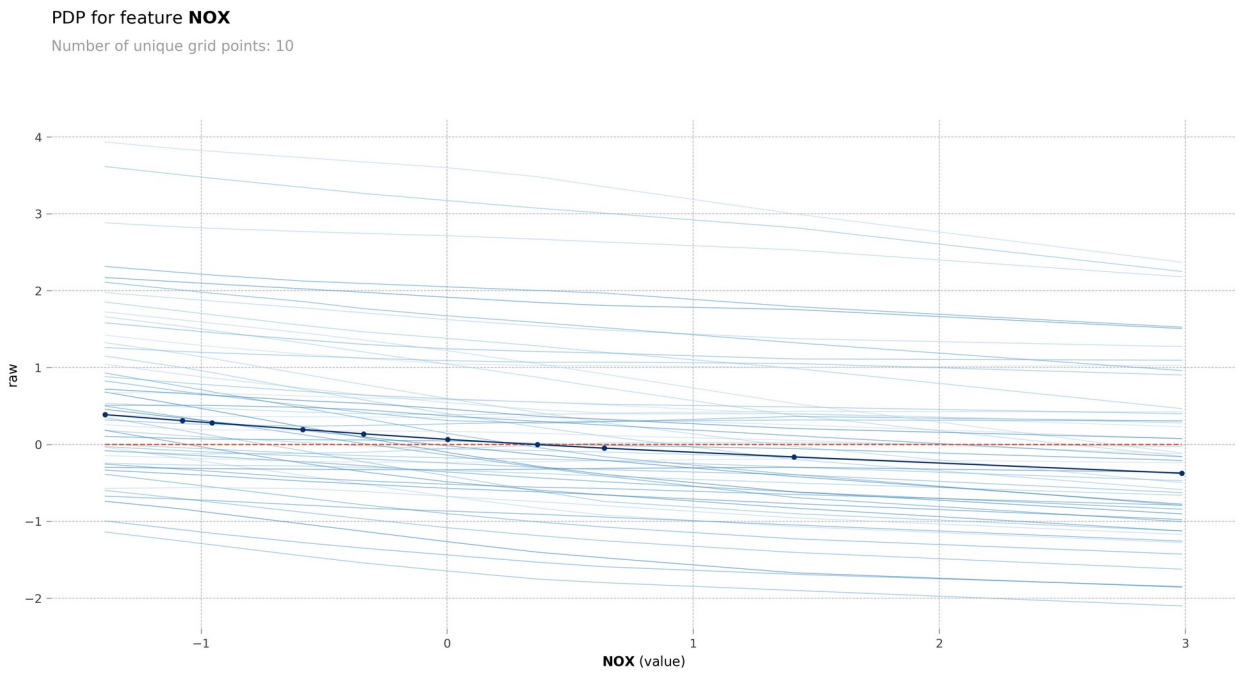

```
obtain pred_func from the provided model.
```


```
{"model_id":"f5dfbed1f25f42a5b09af1541ff0f590","version_major":2,"version_minor":0}
```

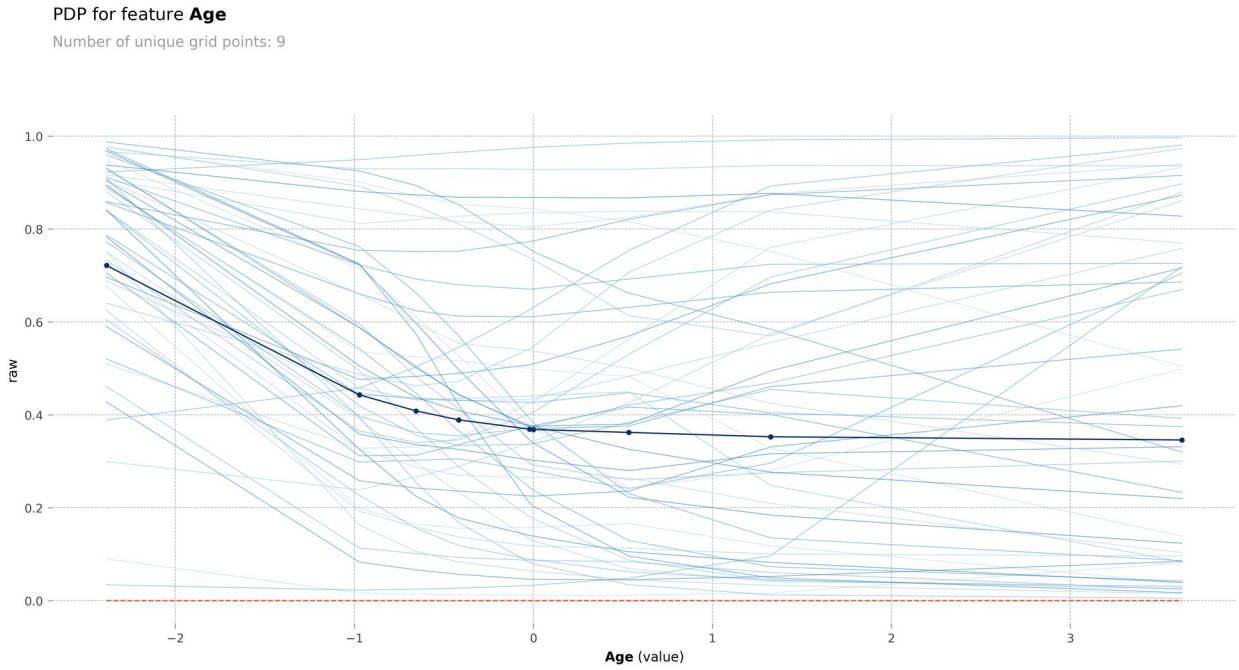

```
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
```



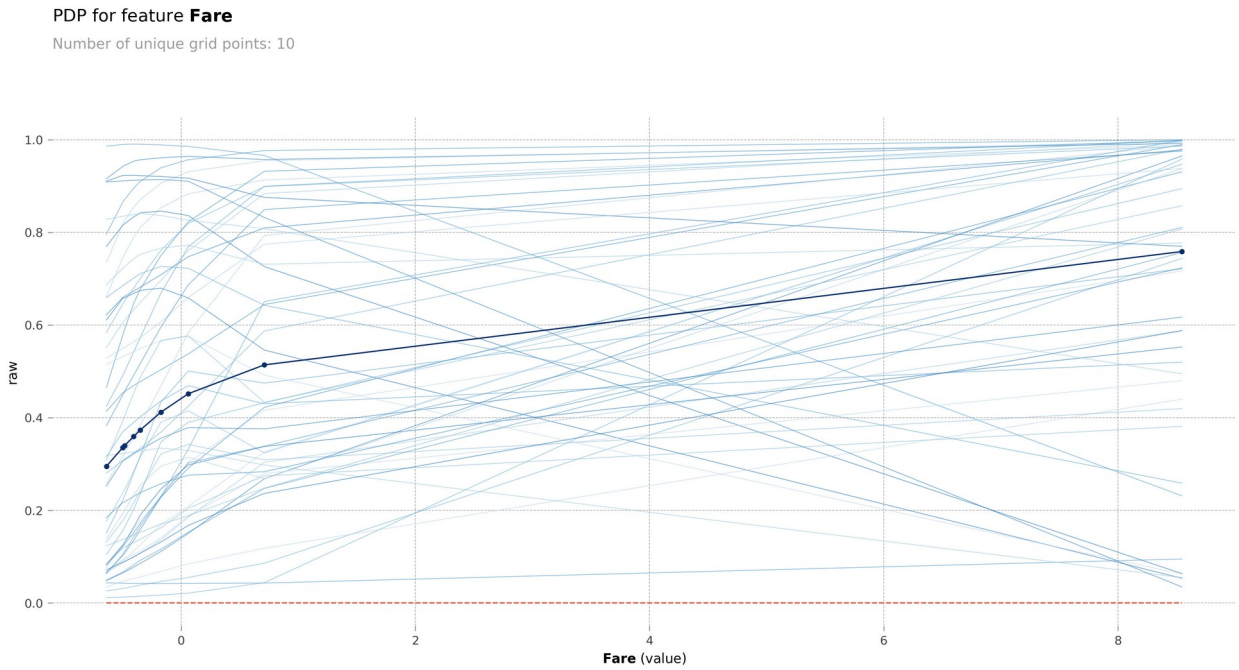

```
obtain pred_func from the provided model.

{"model_id":"990e3d1e80f9478e9d7a2b9a4ed25b18","version_major":2,"vers
ion_minor":0}

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```



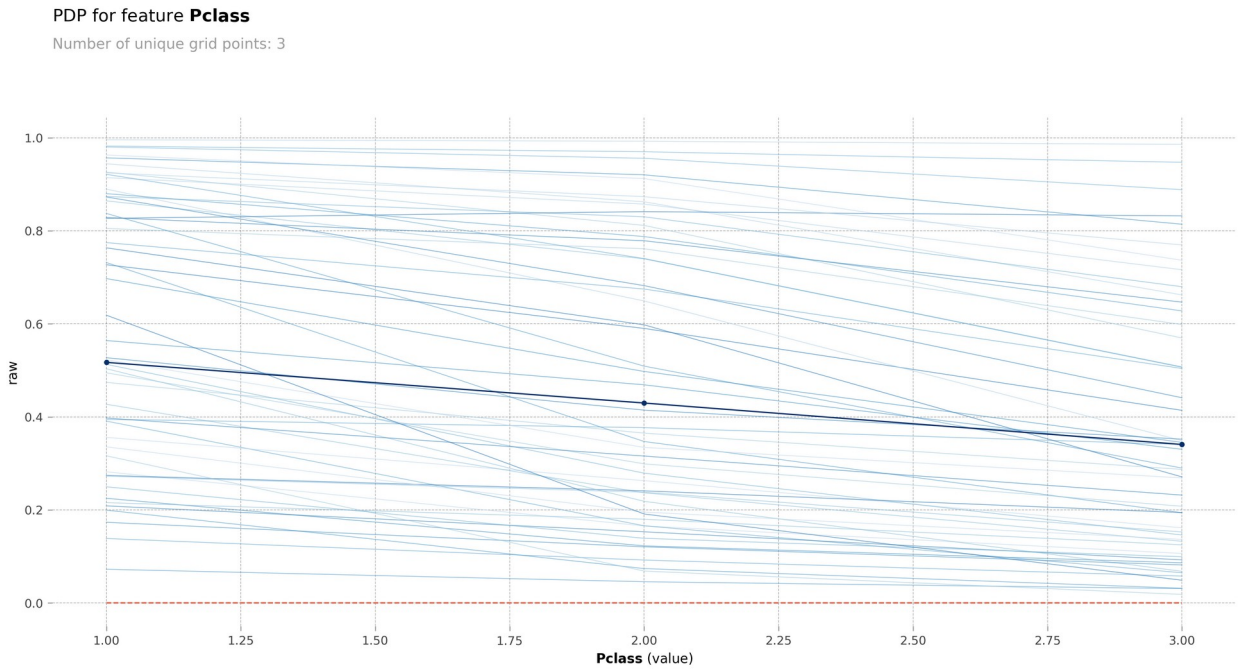

```
obtain pred_func from the provided model.

{"model_id":"e815eb9e816040d4ba10f256c0c10e08","version_major":2,"vers
ion_minor":0}

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```




```
obtain pred_func from the provided model.

{"model_id":"91da0d7031eb45ee93d216ea416954d6","version_major":2,"vers
ion_minor":0}

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```




```
obtain pred_func from the provided model.

{"model_id":"228efb44c954490a9921505c1903c733","version_major":2,"vers
ion_minor":0}

WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not
found.
```



The PDP and ICE plots for the Boston Housing dataset provide crucial insights into the model's understanding of relationships between features and the predicted housing prices. For the feature NOX (nitric oxide concentration), the Partial Dependence Plot indicates a clear negative relationship, where increasing levels of NOX are associated with decreasing predicted housing prices. This reflects the intuition that higher pollution levels correlate with less desirable housing locations, leading to lower property values. The PDP for LSTAT (percentage of lower-status population) similarly shows a strong negative trend, where higher values of LSTAT lead to a sharp decline in predicted housing prices, which aligns with socioeconomic factors impacting neighborhood desirability and housing demand. In contrast, the PDP for the feature B (proportion of Black population by town) shows a slight positive relationship, though the effect is less pronounced compared to NOX and LSTAT.

The ICE plots for the Boston dataset reveal more nuanced individual-level interactions. For NOX, the ICE plot confirms the overall downward trend observed in the PDP but shows that the rate of decline varies for different instances, suggesting the influence of other features like location or socioeconomic metrics. Similarly, the ICE plot for LSTAT highlights consistent negative effects across individual predictions, but some cases exhibit a steeper drop compared to others, reflecting the compounded influence of additional features such as crime rates or school quality. The feature B, while showing a generally positive trend in its PDP, has ICE curves with significant variability. Some individual cases show a much stronger increase in predicted prices as B increases, while others remain relatively stable, suggesting that the effect of this feature is highly context-dependent and influenced by interactions with other factors.

The PDP and ICE plots for the Titanic dataset:

The Partial Dependence Plot (PDP) for the feature Pclass reveals a negative relationship between passenger class and survival probability, where passengers in lower classes (e.g., first-class) have higher survival probabilities compared to those in higher classes (e.g., third-class).

This reflects the historical bias of the Titanic disaster, where wealthier passengers were more likely to survive. Similarly, the PDP for Fare shows a positive relationship, indicating that higher fares, which are often associated with better accommodations, correspond to increased survival probabilities. For the feature Age, the PDP demonstrates a negative trend, where younger passengers are generally more likely to survive, aligning with the prioritization of children during the evacuation.

The ICE plots, however, offer deeper insights by illustrating the individual-level variability in predictions. For Pclass, while the average trend is negative as shown by the PDP, the ICE plot reveals significant variation across individual passengers. Some passengers in higher classes still have low survival probabilities, likely due to interactions with other features such as gender or age. The ICE plot for Fare confirms the positive relationship observed in the PDP, but the varying slopes highlight that for some individuals, an increase in fare leads to a much sharper rise in survival probability. This suggests the influence of correlated features like Pclass. Lastly, the ICE plot for Age shows that while the overall trend is negative, individual predictions vary, with some passengers maintaining stable survival probabilities despite changes in age, suggesting complex interactions with other features.

The analyses of the Titanic and Boston Housing datasets using PDP and ICE plots demonstrate how these interpretability tools can uncover both average feature effects and individual-level nuances in model predictions. In the Titanic dataset, features like Pclass, Fare, and Age exhibit clear average trends that align with historical realities, but ICE plots reveal the variability caused by interactions with other features like gender or socioeconomic status. Similarly, in the Boston Housing dataset, features like NOX and LSTAT show strong average relationships with housing prices, reflecting expected environmental and socioeconomic impacts, while ICE plots highlight the influence of additional interactions and contextual factors. Together, these methods provide a comprehensive understanding of model behavior, ensuring both global interpretability and individualized prediction insights.

2. Permutation Feature Importance (PFI)

```
def calculate_pfi(model_template, X_train, y_train, feature_names,
is_classification=False, n_repeats=5, validation_split=0.2):
    """
    Calculate PFI by retraining model for each permutation and using
validation performance

    Args:
        model_template: untrained model architecture
        X_train: training feature matrix
        y_train: training target values
        feature_names: list of feature names
        is_classification: whether this is a classification task
        n_repeats: number of times to repeat permutation
        validation_split: proportion of data to use for validation
    """
    # Train baseline model
    baseline_model = tf.keras.models.clone_model(model_template)

    if is_classification:
```

```
        baseline_model.compile(
            optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['acc']
        )
    else:
        baseline_model.compile(
            optimizer='adam',
            loss='mean_squared_error',
            metrics=['mean_squared_error', 'mean_absolute_error']
        )

    early_stop = EarlyStopping(
        monitor='val_loss',
        patience=20,
        restore_best_weights=True
    )

    baseline_history = baseline_model.fit(
        X_train, y_train,
        epochs=500,
        validation_split=validation_split,
        batch_size=32 if is_classification else None,
        callbacks=[early_stop],
        verbose=0
    )

    baseline_score = min(baseline_history.history['val_loss'])
    importances = []

    print(f"Baseline validation loss: {baseline_score:.4f}")

    for feature_idx, feature_name in enumerate(feature_names):
        print(f"\nCalculating importance for feature: {feature_name}")
        scores = []

        for repeat in range(n_repeats):
            print(f"Repeat {repeat + 1}/{n_repeats}")

            # Create copy of training data
            X_permuted = X_train.copy()

            # Permute single feature
            X_permuted[:, feature_idx] =
np.random.permutation(X_permuted[:, feature_idx])

            # Create and train new model
            permuted_model =
tf.keras.models.clone_model(model_template)
```

```
            if is_classification:
                permuted_model.compile(
                    optimizer='adam',
                    loss='categorical_crossentropy',
                    metrics=['acc']
                )
            else:
                permuted_model.compile(
                    optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['mean_squared_error',
'mean_absolute_error']
                )

            permuted_history = permuted_model.fit(
                X_permuted, y_train,
                epochs=500,
                validation_split=validation_split,
                batch_size=32 if is_classification else None,
                callbacks=[early_stop],
                verbose=0
            )

            # Get best validation loss
            permuted_score = min(permuted_history.history['val_loss'])

            # Calculate importance (increase in loss)
            importance = permuted_score - baseline_score
            scores.append(importance)

        mean_importance = np.mean(scores)
        std_importance = np.std(scores)
        importances.append((feature_name, mean_importance,
std_importance))

    importance_df = pd.DataFrame(importances, columns=['Feature',
'Importance', 'Std'])
    importance_df = importance_df.sort_values('Importance',
ascending=False)

    return importance_df

# Calculate PFI for Boston Housing model
boston_model_template = keras.Sequential([
  keras.layers.Dense(32, activation = 'relu', input_shape = (13,)),
  keras.layers.Dropout(0.1),
  keras.layers.Dense(1)
])

# Calculate PFI for Boston Housing model
```

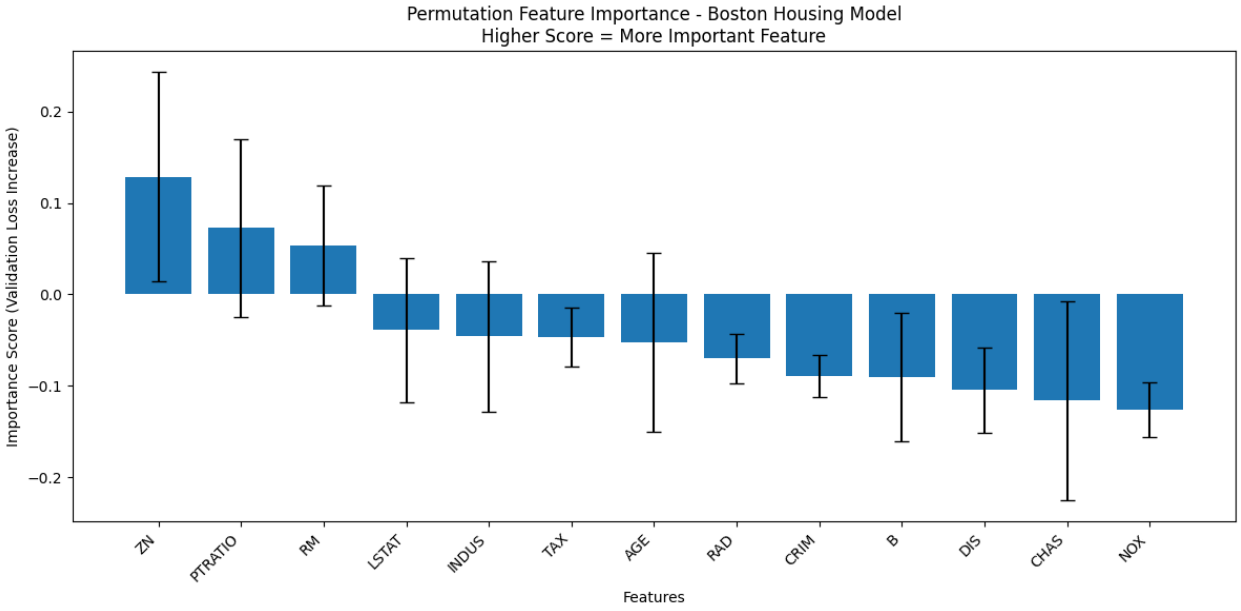
```
# use test set and test lables for boston
boston_pfi = calculate_pfi(
    boston_model,  # Use trained model
    boston_test_features_scaled_df.values,  # Use test set
    boston_test_labels_scaled,
    boston_feature_names,
    is_classification=False
)

# Calculate PFI for Titanic model
# use train set for titanic as kaggle train set does not have labels
available
titanic_model_template = keras.Sequential([
  tf.keras.layers.Dense(32, activation = 'relu'),
  tf.keras.layers.Dropout(0.1),
  tf.keras.layers.Dense(2, activation = 'softmax')
])

titanic_pfi = calculate_pfi(
    titanic_model_template,
    titanic_train_features.values,
    titanic_train_labels,
    titanic_train_features.columns,
    is_classification=True
)

# Visualization function
def plot_feature_importance(importance_df, title):
    plt.figure(figsize=(12, 6))

    plt.bar(
        importance_df['Feature'],
        importance_df['Importance'],
        yerr=importance_df['Std'],
        capsize=5
    )

    plt.xticks(rotation=45, ha='right')
    plt.xlabel('Features')
    plt.ylabel('Importance Score (Validation Loss Increase)')
    plt.title(f'Permutation Feature Importance - {title}\nHigher Score
= More Important Feature')
    plt.tight_layout()
    plt.show()

    print(f"\nFeature Importance Rankings - {title}:")
    print(importance_df.to_string(index=False))

# Plot results for both models
```

```
plot_feature_importance(boston_pfi, "Boston Housing Model")
plot_feature_importance(titanic_pfi, "Titanic Model")
```

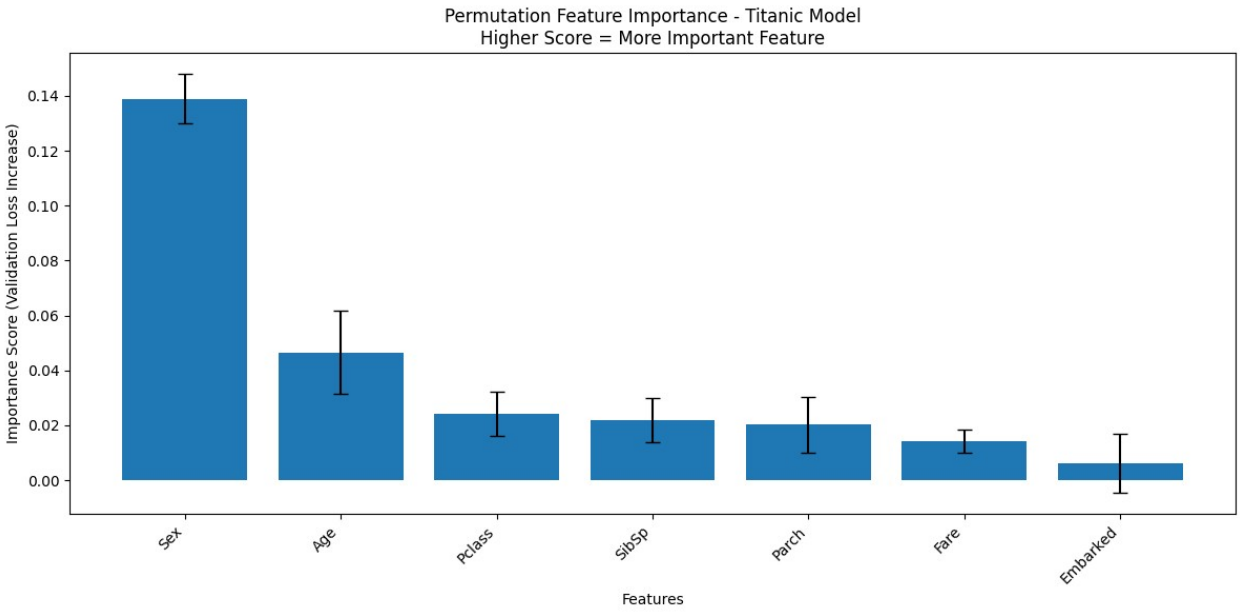
```
Baseline validation loss: 0.4405

Calculating importance for feature: CRIM
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: ZN
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: INDUS
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: CHAS
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: NOX
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: RM
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: AGE
Repeat 1/5
Repeat 2/5
```

```
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: DIS
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: RAD
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: TAX
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: PTRATIO
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: B
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: LSTAT
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5
Baseline validation loss: 0.3394

Calculating importance for feature: Pclass
Repeat 1/5
Repeat 2/5
```

```
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: Sex
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: Age
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: SibSp
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: Parch
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: Fare
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5

Calculating importance for feature: Embarked
Repeat 1/5
Repeat 2/5
Repeat 3/5
Repeat 4/5
Repeat 5/5
```




```
Feature Importance Rankings - Boston Housing Model:
Feature  Importance      Std
     ZN    0.128712 0.114293
PTRATIO    0.072572 0.097738
     RM    0.053625 0.065898
  LSTAT   -0.039162 0.078932
  INDUS   -0.046090 0.082749
    TAX   -0.046980 0.032247
    AGE   -0.052166 0.098002
    RAD   -0.070007 0.027156
   CRIM   -0.089513 0.022986
      B   -0.090543 0.069809
    DIS   -0.104771 0.046608
   CHAS   -0.116236 0.108631
    NOX   -0.126033 0.030012
```




```
Feature Importance Rankings - Titanic Model:
 Feature  Importance      Std
     Sex    0.139020 0.009017
     Age    0.046602 0.015045
  Pclass    0.024189 0.007981
   SibSp    0.021940 0.008110
   Parch    0.020173 0.010210
    Fare    0.014198 0.004344
Embarked    0.006062 0.010675
```

Titanic Model:

The most important feature in the Titanic model, as determined by Permutation Feature Importance (PFI), is Sex with an importance value of 0.139020 and a standard deviation of 0.009017. This is followed by Age (0.046602) and Pclass (0.024189), both of which also have notable contributions, albeit to a lesser extent.

Boston Housing Model:

In the Boston Housing model, the most important feature is ZN (proportion of residential land zoned for large lots) with an importance value of 0.128712 and a standard deviation of 0.114293. Other features with positive contributions include PTRATIO (pupil-teacher ratio) with 0.072572 and RM (average number of rooms per dwelling) with 0.053625. It is noteworthy that many features, such as NOX and DIS, have negative importance values, indicating potential non-linear interactions or the model relying on these features in counterintuitive ways.

In the context of PFI, "importance" refers to how much the model's performance decreases when the values of a specific feature are randomly permuted. This method measures the contribution of each feature to the model's predictive accuracy by introducing noise and observing how the model's predictions are affected.

A feature is considered important if permuting it significantly degrades the model's performance, meaning that the model heavily relies on that feature to make accurate predictions. Conversely, a feature with low or negative importance has little to no influence on the model's decisions or might even introduce noise into the predictions. For example, in the Titanic model, the high importance of Sex indicates that survival predictions rely heavily on this feature, reflecting real-world evacuation patterns. In the Boston Housing model, ZN is deemed most important because it significantly impacts housing price predictions, likely due to its association with suburban zoning and property value.

In summary, PFI quantifies the dependency of the model on each feature by testing how predictions deteriorate when that feature's information is disrupted.

Accumulated Local Effects (ALE)

```
def calculate_ale(model, X, feature_name, num_bins=20,
is_categorical=False, is_classification=False):
    """
    Calculate Accumulated Local Effects with proper categorical
handling
    """
    feature_data = X[feature_name].values

    if is_categorical:
        # For categorical variables, use unique values directly
        unique_values = np.sort(np.unique(feature_data))
        ale_values = np.zeros(len(unique_values))

        # Calculate effect for each category
        for i, value in enumerate(unique_values):
            instances = X.copy()
            instances[feature_name] = value
            predictions = model.predict(instances.values)

            if is_classification:
                predictions = predictions[:, 1]

            ale_values[i] = np.mean(predictions)

        # Center ALE values
        ale_values -= np.mean(ale_values)
        return unique_values, ale_values
    else:
        # For continuous variables, use quantile binning
        bins = np.percentile(feature_data, np.linspace(0, 100,
num_bins))
        bins = np.unique(bins)
        num_bins = len(bins) - 1
        ale_values = np.zeros(num_bins)

        for i in range(num_bins):
            mask = (feature_data >= bins[i]) & (feature_data <= bins[i
```

```
+ 1])
            if not np.any(mask):
                continue

            instances = X[mask].copy()

            # Calculate effects
            x_lower = instances.copy()
            x_upper = instances.copy()
            x_lower[feature_name] = bins[i]
            x_upper[feature_name] = bins[i + 1]

            pred_lower = model.predict(x_lower.values)
            pred_upper = model.predict(x_upper.values)

            if is_classification:
                pred_lower = pred_lower[:, 1]
                pred_upper = pred_upper[:, 1]

            ale_values[i] = np.mean(pred_upper - pred_lower)

        # Accumulate and center
        ale_values = np.cumsum(ale_values)
        ale_values -= np.mean(ale_values)
        return bins, ale_values

def plot_ale_all_features(model, X, title, categorical_features=[],
is_classification=False):
    """
    Plot ALE for all features with proper categorical handling
    """
    n_features = X.shape[1]
    n_cols = 3
    n_rows = (n_features + n_cols - 1) // n_cols

    fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))
    axes = axes.ravel()

    importance_scores = []

    for i, feature_name in enumerate(X.columns):
        print(f"Processing feature: {feature_name}")
        is_categorical = feature_name in categorical_features
        values, ale_values = calculate_ale(
            model,
            X,
            feature_name,
            is_categorical=is_categorical,
            is_classification=is_classification
        )
```

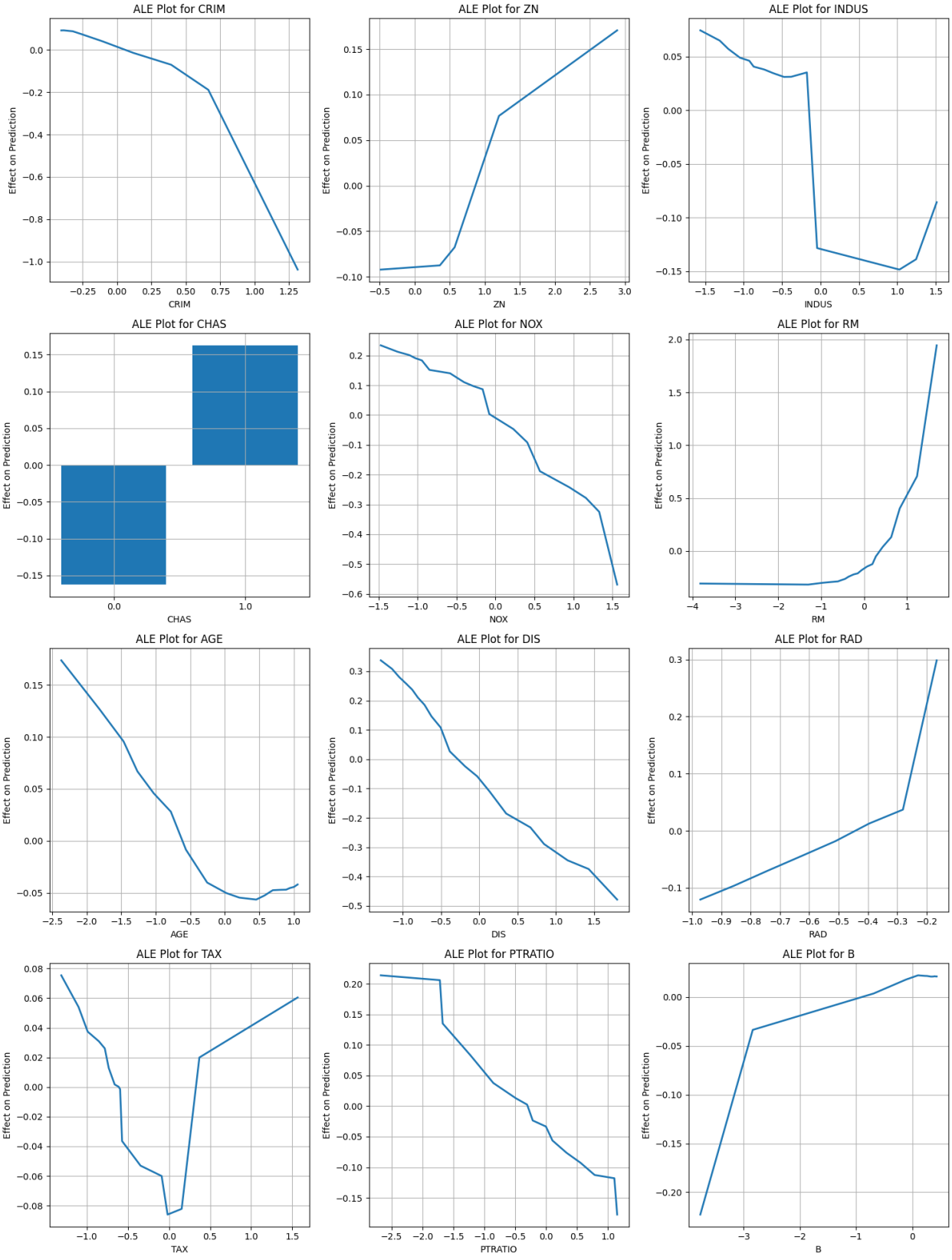
```
        if is_categorical:
            # For categorical features, use bar plot
            axes[i].bar(range(len(values)), ale_values)
            axes[i].set_xticks(range(len(values)))
            axes[i].set_xticklabels(values)
        else:
            # For continuous features, use line plot
            axes[i].plot(values[:-1], ale_values, '-', linewidth=2)

        axes[i].set_title(f'ALE Plot for {feature_name}')
        axes[i].set_xlabel(feature_name)
        axes[i].set_ylabel('Effect on Prediction')
        axes[i].grid(True)

        importance = np.max(ale_values) - np.min(ale_values)
        importance_scores.append((feature_name, importance))

    for i in range(n_features, len(axes)):
        fig.delaxes(axes[i])

    plt.suptitle(f'ALE Plots for {title}', y=1.02, fontsize=16)
    plt.tight_layout()
    plt.show()

    importance_scores.sort(key=lambda x: x[1], reverse=True)
    print(f"\nFeature Importance based on ALE range for {title}:")
    for feature, importance in importance_scores:
        print(f"{feature}: {importance:.4f}")

def plot_ale_interaction(model, X, feature1, feature2, num_bins=8,
is_classification=False):
    """
    Plot 2D ALE for feature interactions
    """
    f1_bins = np.percentile(X[feature1], np.linspace(0, 100, num_bins
+ 1))
    f2_bins = np.percentile(X[feature2], np.linspace(0, 100, num_bins
+ 1))

    ale_matrix = np.zeros((num_bins, num_bins))

    for i in range(num_bins):
        print(f"Processing interaction row {i+1}/{num_bins}")
        for j in range(num_bins):
            mask = (
                (X[feature1] >= f1_bins[i]) &
                (X[feature1] <= f1_bins[i + 1]) &
                (X[feature2] >= f2_bins[j]) &
                (X[feature2] <= f2_bins[j + 1])
```

```
            )

            instances = X[mask]
            if len(instances) == 0:
                continue

            predictions = model.predict(instances.values)
            if is_classification:
                predictions = predictions[:, 1]

            ale_matrix[i, j] = np.mean(predictions)

    plt.figure(figsize=(10, 8))
    plt.imshow(ale_matrix, cmap='RdBu', aspect='auto')
    plt.colorbar(label='ALE value')
    plt.title(f'ALE Interaction: {feature1} vs {feature2}')
    plt.xlabel(feature1)
    plt.ylabel(feature2)
    plt.show()

# Run analysis for Boston Housing Dataset
print("ALE Analysis for Boston Housing Dataset:")
plot_ale_all_features(
    boston_model,
    boston_train_features_scaled_df,
    "Boston Housing Dataset",
    categorical_features=['CHAS'],
    is_classification=False
)

# Run analysis for Titanic Dataset
print("\nALE Analysis for Titanic Dataset:")
plot_ale_all_features(
    titanic_model,
    titanic_train_features,
    "Titanic Dataset",
    categorical_features=['Sex', 'Embarked'],
    is_classification=True
)

# Plot feature interactions for Boston Housing Dataset
print("\nFeature Interactions for Boston Housing Dataset:")
plot_ale_interaction(boston_model, boston_train_features_scaled_df,
'LSTAT', 'RM', is_classification=False)
plot_ale_interaction(boston_model, boston_train_features_scaled_df,
'NOX', 'DIS', is_classification=False)

# Plot feature interactions for Titanic Dataset
print("\nFeature Interactions for Titanic Dataset:")
plot_ale_interaction(titanic_model, titanic_train_features, 'Age',
```

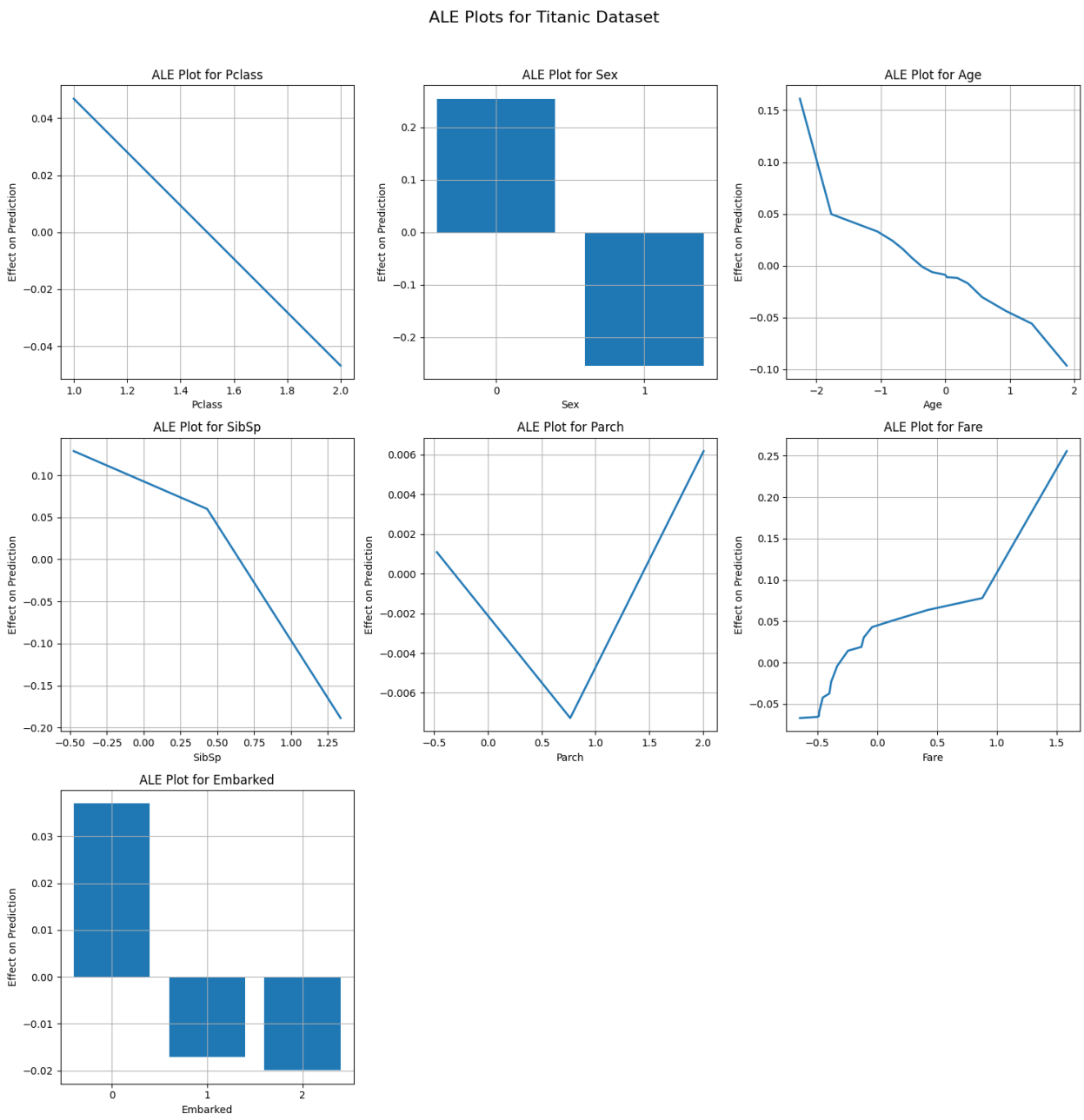
```
'Fare', is_classification=True)
plot_ale_interaction(titanic_model, titanic_train_features, 'Pclass',
'Age', is_classification=True)
```

```
ALE Analysis for Boston Housing Dataset:
Processing feature: CRIM
Processing feature: ZN
Processing feature: INDUS
Processing feature: CHAS
Processing feature: NOX
Processing feature: RM
Processing feature: AGE
Processing feature: DIS
Processing feature: RAD
Processing feature: TAX
Processing feature: PTRATIO
Processing feature: B
Processing feature: LSTAT
```

ALE Plots for Boston Housing Dataset

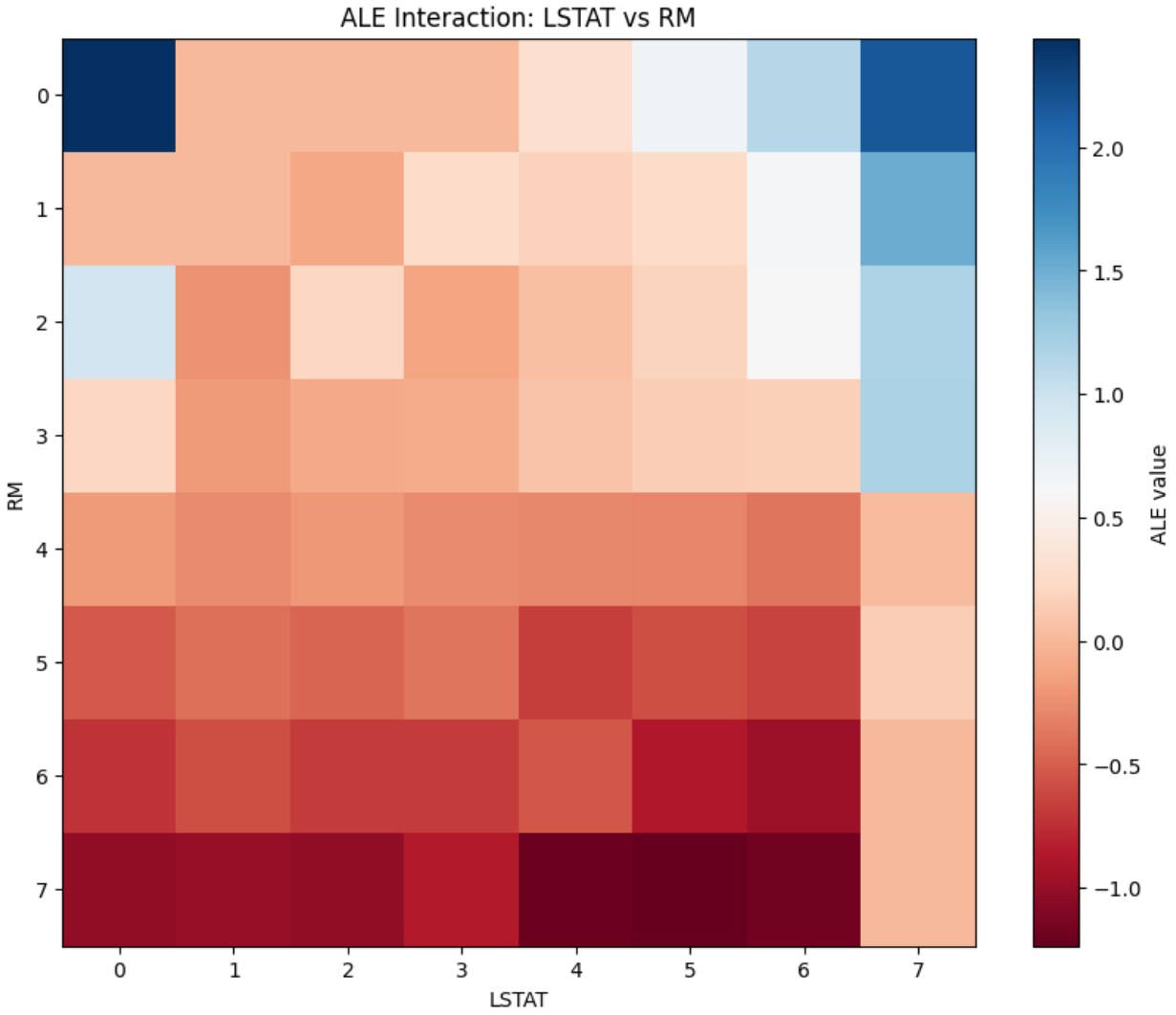


```
Feature Importance based on ALE range for Boston Housing Dataset:
RM: 2.2616
LSTAT: 1.6907
CRIM: 1.1305
DIS: 0.8159
NOX: 0.8025
RAD: 0.4190
PTRATIO: 0.3911
CHAS: 0.3251
ZN: 0.2628
B: 0.2451
AGE: 0.2303
INDUS: 0.2227
TAX: 0.1615

ALE Analysis for Titanic Dataset:
Processing feature: Pclass
Processing feature: Sex
Processing feature: Age
Processing feature: SibSp
Processing feature: Parch
Processing feature: Fare
Processing feature: Embarked
```



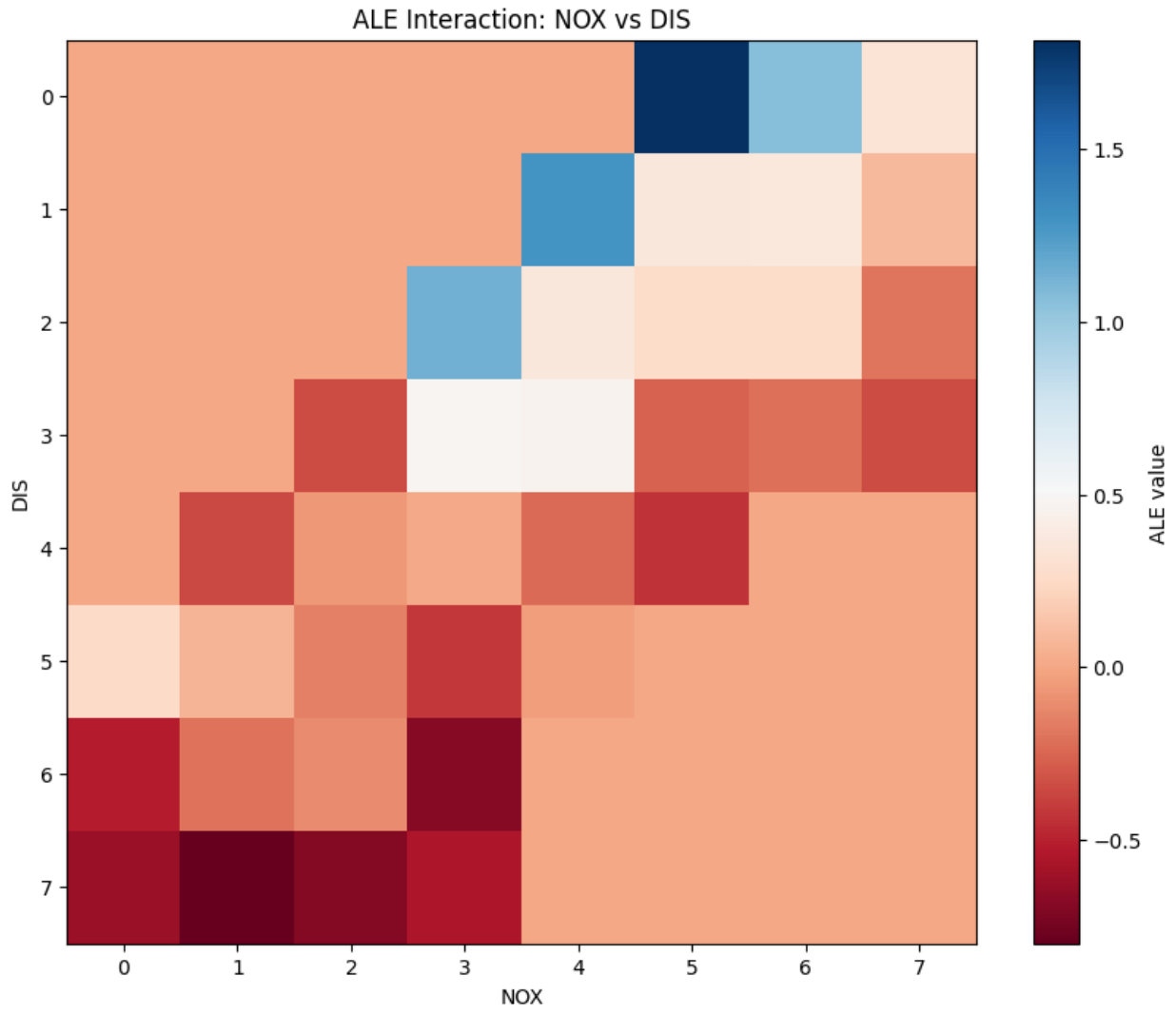

```
Feature Importance based on ALE range for Titanic Dataset:
Sex: 0.5082
Fare: 0.3225
SibSp: 0.3172
Age: 0.2577
Pclass: 0.0937
Embarked: 0.0569
Parch: 0.0135

Feature Interactions for Boston Housing Dataset:
```

```
Processing interaction row 1/8
Processing interaction row 2/8
Processing interaction row 3/8
Processing interaction row 4/8
Processing interaction row 5/8
Processing interaction row 6/8
Processing interaction row 7/8
Processing interaction row 8/8
```

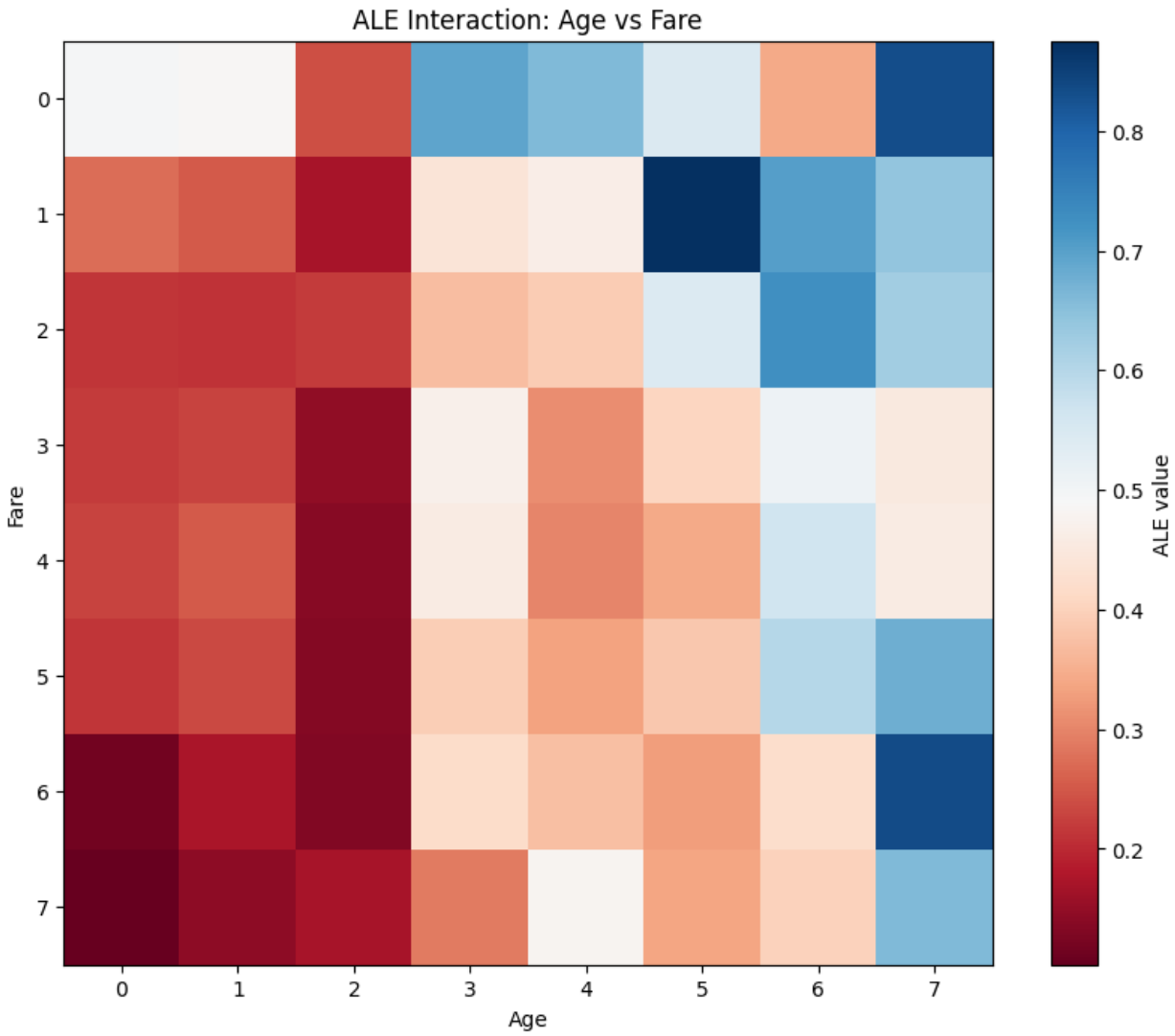



```
Processing interaction row 1/8
Processing interaction row 2/8
Processing interaction row 3/8
Processing interaction row 4/8
Processing interaction row 5/8
Processing interaction row 6/8
```

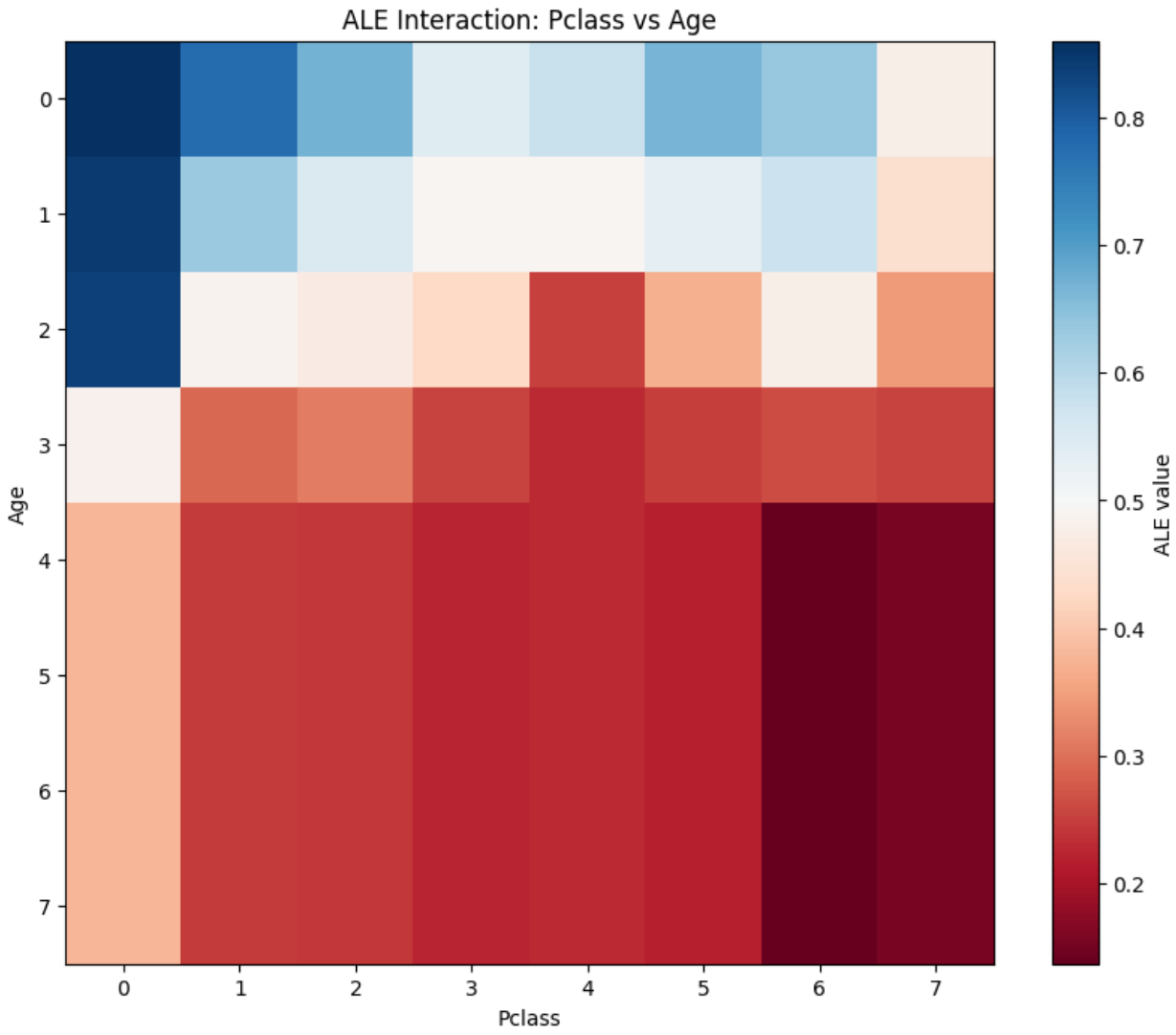
```
Processing interaction row 7/8
Processing interaction row 8/8
```




```
Feature Interactions for Titanic Dataset:
Processing interaction row 1/8
Processing interaction row 2/8
Processing interaction row 3/8
Processing interaction row 4/8
Processing interaction row 5/8
Processing interaction row 6/8
Processing interaction row 7/8
Processing interaction row 8/8
```




```
Processing interaction row 1/8
Processing interaction row 2/8
Processing interaction row 3/8
Processing interaction row 4/8
Processing interaction row 5/8
Processing interaction row 6/8
Processing interaction row 7/8
Processing interaction row 8/8
```



ALE Insights:

The ALE plots provide a nuanced understanding of the local effects of feature changes on predictions for both datasets. For the Titanic dataset, Sex has the highest ALE range (0.5082), indicating it has the strongest local effect on the model's predictions, aligning with societal patterns during the Titanic disaster where women were prioritized for survival. The Fare feature also has a notable effect (0.3225), with higher fare values significantly increasing survival probabilities, reflecting the association of wealth with better survival outcomes. Features like SibSp and Age also show substantial contributions, with decreasing age leading to higher survival probabilities, consistent with the prioritization of children.

In the Boston Housing dataset, RM (average number of rooms per dwelling) has the largest ALE range (2.2616), demonstrating a strong local effect where more rooms lead to higher housing prices. This aligns with the expectation that spacious homes are more valuable. Similarly, LSTAT (percentage of lower-status population) exhibits a high negative ALE range (1.6907), suggesting that higher proportions of lower-status individuals in a neighborhood lead to significant drops in predicted housing prices. Features such as CRIM (crime rate) and DIS (distance to employment

centers) also show meaningful effects, with higher crime rates and greater distances reducing housing prices.

Comparison of ALE and PDP:

While both ALE and PDP analyze feature effects, they differ in key aspects of interpretability. PDPs measure the average marginal effect of features on predictions across the entire dataset, assuming independence from other features. For instance, the PDPs for the Titanic dataset showed a strong average trend for Sex and Fare in survival predictions and for RM and LSTAT in housing prices. However, PDPs can suffer from biases caused by correlated features, as they fail to disentangle interaction effects.

ALE plots, on the other hand, are conditional and account for interactions between features, providing a more accurate depiction of localized effects. For example, in the Titanic dataset, the ALE plot for Pclass shows less drastic effects compared to its PDP counterpart, reflecting its conditional role when accounting for Fare and other features. Similarly, in the Boston Housing dataset, the ALE plot for LSTAT highlights more nuanced non-linear effects on predictions, which are not as apparent in the PDP. ALE also avoids extrapolation issues by focusing on the range of observed feature values.

In summary, PDPs provide a broader view of feature effects, but ALE plots offer more robust and localized insights by incorporating feature interactions and avoiding assumptions of independence. Together, these tools complement each other, with PDPs helping to understand global trends and ALE enhancing the interpretability of feature-specific local effects.

Global Surrogates

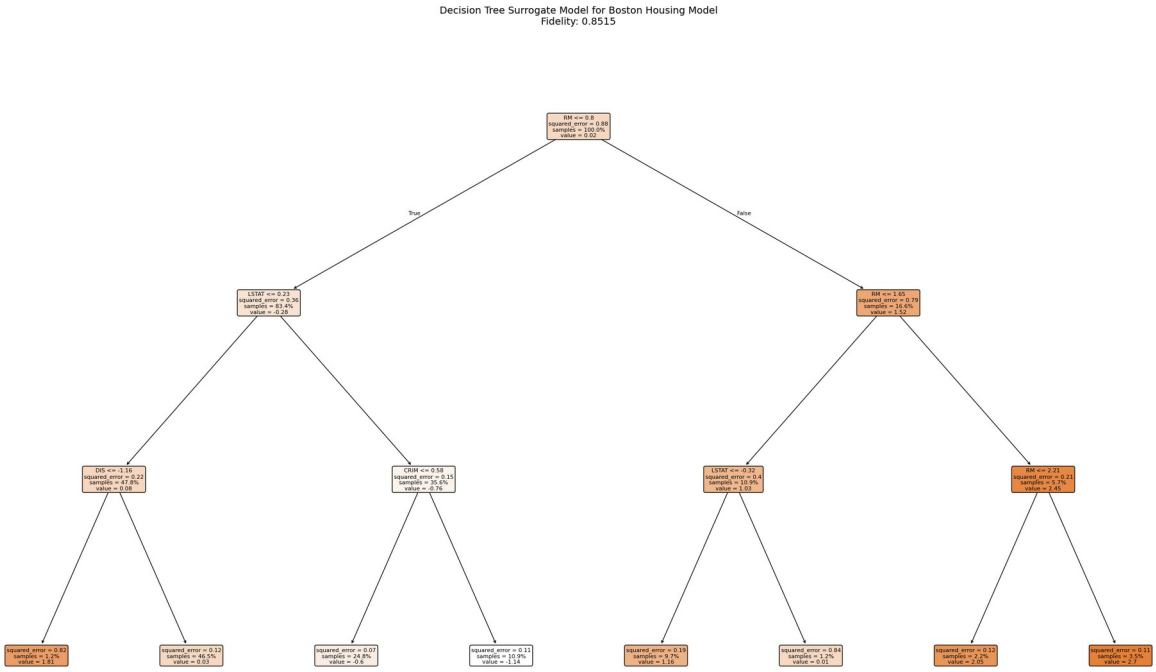
```
# Function to train and evaluate surrogate models
def train_surrogate_model(X, model, is_classification=False):
    """
    Train a decision tree surrogate model to approximate the neural
network
    """
    predictions = model.predict(X)
    if is_classification:
        predictions = np.argmax(predictions, axis=1)
        surrogate = DecisionTreeClassifier(
            max_depth=3,
            min_samples_leaf=5,
            random_state=42
        )
        metric = accuracy_score
    else:
        surrogate = DecisionTreeRegressor(
            max_depth=3,
            min_samples_leaf=5,
            random_state=42
        )
        metric = r2_score

    surrogate.fit(X, predictions)
```

```
    surrogate_predictions = surrogate.predict(X)
    fidelity = metric(predictions, surrogate_predictions)

    return surrogate, fidelity

# Train surrogate for Boston Housing model
boston_surrogate, boston_fidelity = train_surrogate_model(
    boston_train_features_scaled_df.values,
    boston_model,
    is_classification=False
)

# Train surrogate for Titanic model
titanic_surrogate, titanic_fidelity = train_surrogate_model(
    titanic_train_features.values,
    titanic_model,
    is_classification=True
)

# Visualization function
def visualize_surrogate(surrogate, feature_names, title, fidelity):
    plt.figure(figsize=(25, 15))
    plot_tree(
        surrogate,
        feature_names=feature_names,
        filled=True,
        rounded=True,
        fontsize=8,
        proportion=True,
        precision=2,
        class_names=['0', '1'] if isinstance(surrogate,
DecisionTreeClassifier) else None
    )
    plt.title(f'Decision Tree Surrogate Model for {title}\nFidelity:
{fidelity:.4f}', fontsize=14, pad=20)
    plt.tight_layout()
    plt.show()

# Feature importance from surrogate models
def print_feature_importance(surrogate, feature_names, title):
    importances = surrogate.feature_importances_
    indices = np.argsort(importances)[::-1]

    print(f"\nFeature Importance from {title} Surrogate Model:")
    for f in range(len(feature_names)):
        print(f"{feature_names[indices[f]]:20s}:
{importances[indices[f]]:.4f}")

    # Bar plot of feature importances
    plt.figure(figsize=(12, 6))
```

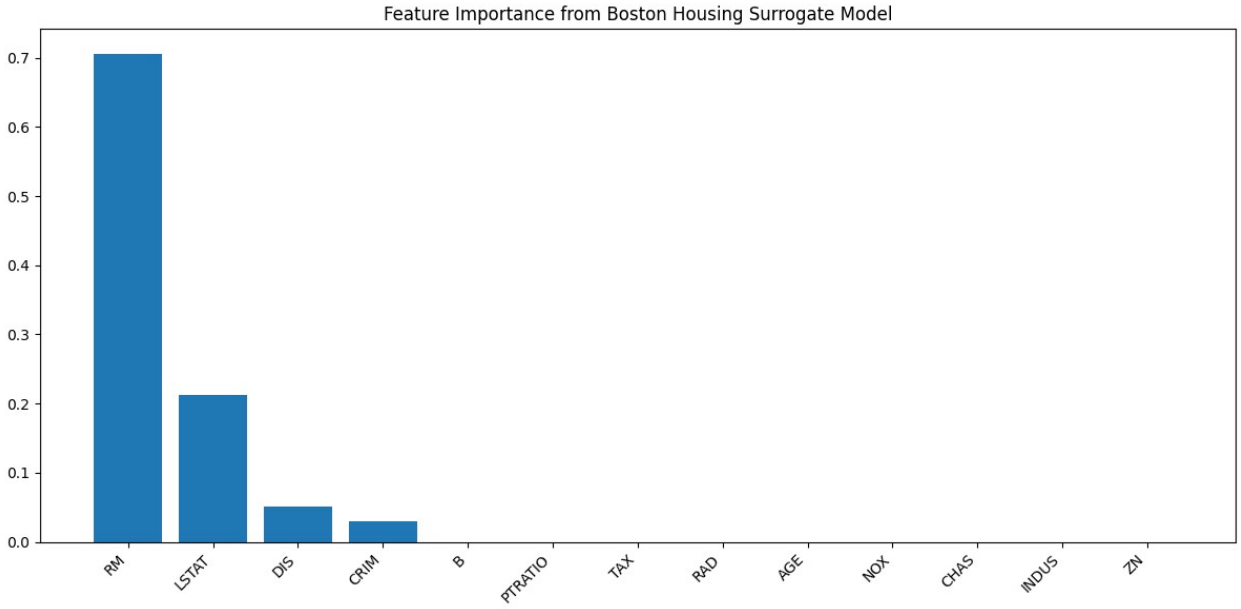
```
    plt.bar(range(len(feature_names)), importances[indices])
    plt.xticks(range(len(feature_names)), [feature_names[i] for i in
indices], rotation=45, ha='right')
    plt.title(f'Feature Importance from {title} Surrogate Model')
    plt.tight_layout()
    plt.show()

# Visualize and analyze Boston Housing surrogate
print("Boston Housing Model Surrogate:")
print(f"Fidelity (R² Score): {boston_fidelity:.4f}")
visualize_surrogate(boston_surrogate, boston_feature_names, "Boston
Housing Model", boston_fidelity)
print_feature_importance(boston_surrogate, boston_feature_names,
"Boston Housing")

# Visualize and analyze Titanic surrogate
print("\nTitanic Model Surrogate:")
print(f"Fidelity (Accuracy): {titanic_fidelity:.4f}")
visualize_surrogate(titanic_surrogate, titanic_train_features.columns,
"Titanic Model", titanic_fidelity)
print_feature_importance(titanic_surrogate,
titanic_train_features.columns, "Titanic")

# Print example predictions comparison
def compare_predictions(surrogate, original_model, X,
is_classification=False, n_samples=5):
    print(f"\nComparing {n_samples} example predictions:")
    original_preds = original_model.predict(X[:n_samples])
    if is_classification:
        original_preds = np.argmax(original_preds, axis=1)
    surrogate_preds = surrogate.predict(X[:n_samples])

    for i in range(n_samples):
        print(f"Sample {i+1}:")
        if is_classification:
            print(f"  Original model prediction: Class
{original_preds[i]}")
            print(f"  Surrogate model prediction: Class
{surrogate_preds[i]}")
        else:
            print(f"  Original model prediction:
{float(original_preds[i]):.4f}")
            print(f"  Surrogate model prediction:
{float(surrogate_preds[i]):.4f}")
        print("  " + "="*40)

# Compare predictions for both models
print("\nBoston Housing Model Predictions:")
compare_predictions(boston_surrogate, boston_model,
boston_train_features_scaled_df.values, is_classification=False)
```

```
print("\nTitanic Model Predictions:")
compare_predictions(titanic_surrogate, titanic_model,
titanic_train_features.values, is_classification=True)
```

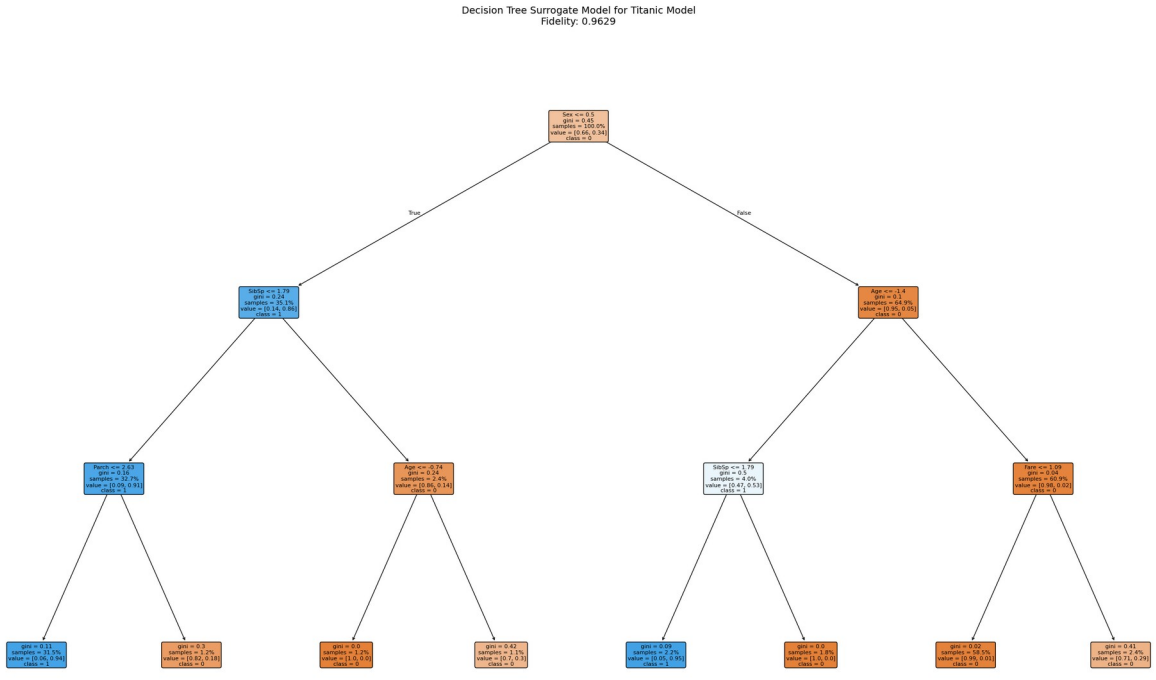
```
Boston Housing Model Surrogate:
Fidelity (R² Score): 0.8515
```




```
Feature Importance from Boston Housing Surrogate Model:
RM                   : 0.7064
LSTAT                : 0.2128
DIS                  : 0.0509
CRIM                 : 0.0299
B                    : 0.0000
PTRATIO              : 0.0000
TAX                  : 0.0000
RAD                  : 0.0000
AGE                  : 0.0000
NOX                  : 0.0000
CHAS                 : 0.0000
INDUS                : 0.0000
ZN                   : 0.0000
```

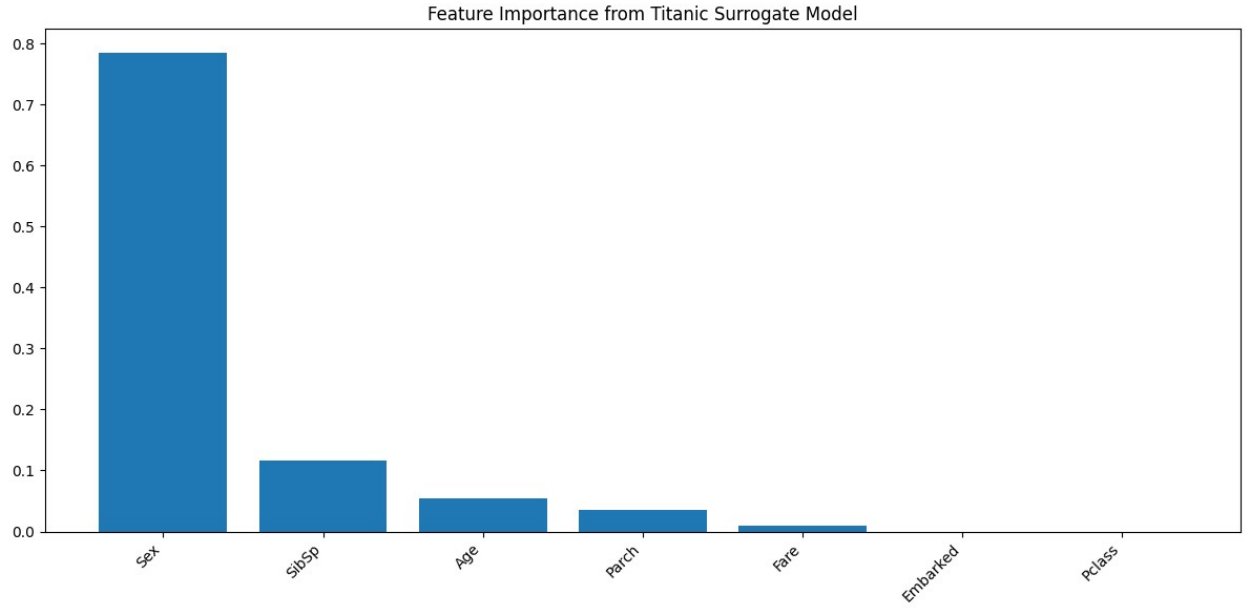



```
Titanic Model Surrogate:
Fidelity (Accuracy): 0.9629
```




```
Feature Importance from Titanic Surrogate Model:
Sex                  : 0.7845
```

```
SibSp                : 0.1161
Age                  : 0.0541
Parch                : 0.0362
Fare                 : 0.0091
Embarked             : 0.0000
Pclass               : 0.0000
```




```
Boston Housing Model Predictions:

Comparing 5 example predictions:
Sample 1:
  Original model prediction: -0.6777
  Surrogate model prediction: -0.5971
  ========================================
Sample 2:
  Original model prediction: 2.1889
  Surrogate model prediction: 2.0541
  ========================================
Sample 3:
  Original model prediction: 2.4420
  Surrogate model prediction: 1.8102
  ========================================
Sample 4:
  Original model prediction: -0.0998
  Surrogate model prediction: 0.0324
  ========================================
Sample 5:
  Original model prediction: -0.3786
  Surrogate model prediction: -0.5971
```

```
  ========================================

Titanic Model Predictions:

Comparing 5 example predictions:
Sample 1:
  Original model prediction: Class 0
  Surrogate model prediction: Class 0
  ========================================
Sample 2:
  Original model prediction: Class 1
  Surrogate model prediction: Class 1
  ========================================
Sample 3:
  Original model prediction: Class 1
  Surrogate model prediction: Class 1
  ========================================
Sample 4:
  Original model prediction: Class 1
  Surrogate model prediction: Class 1
  ========================================
Sample 5:
  Original model prediction: Class 0
  Surrogate model prediction: Class 0
  ========================================
```

Analysis of Surrogate Models

For the Boston Housing model, the decision tree surrogate achieved a fidelity score (R^2) of 0.8515, indicating that the surrogate model effectively approximates the predictions of the neural network. The surrogate's feature importance highlights that RM (average number of rooms) is the most critical factor (importance: 0.7064) influencing housing price predictions, followed by LSTAT (percentage of lower-status population) with an importance of 0.2128. Other features, such as DIS (distance to employment centers) and CRIM (crime rate), have minimal contributions, while the remaining features are not utilized by the surrogate. This aligns well with insights from the neural network's interpretability techniques like PDP, ALE, and PFI, where RM and LSTAT consistently emerged as key predictors.

For the Titanic model, the decision tree surrogate achieved a high fidelity score of 0.9629, demonstrating that the surrogate closely matches the neural network's predictions. The surrogate model identifies Sex as the dominant predictor (importance: 0.7845), consistent with prior findings across other methods. SibSp (number of siblings/spouses aboard) and Age also have notable contributions (importance: 0.1161 and 0.0541, respectively), while other features like Parch, Fare, and Embarked play minimal roles. Notably, Pclass, which was important in previous analyses, does not contribute directly in the surrogate model, potentially because its effects are indirectly represented through other correlated features.

Effectiveness and Utility of Surrogate Models

Surrogate models are valuable tools for approximating complex, opaque models like neural networks with more interpretable structures such as decision trees. Their utility lies in their ability to provide a global overview of feature importance and decision-making processes, offering insights into how predictions are generated. In this case, the Boston Housing surrogate provides a straightforward interpretation of the neural network's reliance on key features like RM and LSTAT, while the Titanic surrogate reveals the overwhelming importance of Sex and the nuanced contributions of SibSp and Age.

These approximations are particularly helpful in scenarios where model transparency is critical, such as regulatory environments, healthcare, and finance. By using surrogates, stakeholders can understand and trust model predictions without requiring in-depth knowledge of neural networks. However, their effectiveness depends on fidelity—low fidelity scores indicate that the surrogate cannot accurately replicate the original model's behavior. Additionally, surrogates may oversimplify complex relationships, so they should be used in conjunction with other interpretability techniques to capture both global and local patterns effectively.

Part 2: Local Interpretability Techniques

1. Local Interpretable Model-agnostic Explanations (LIME)

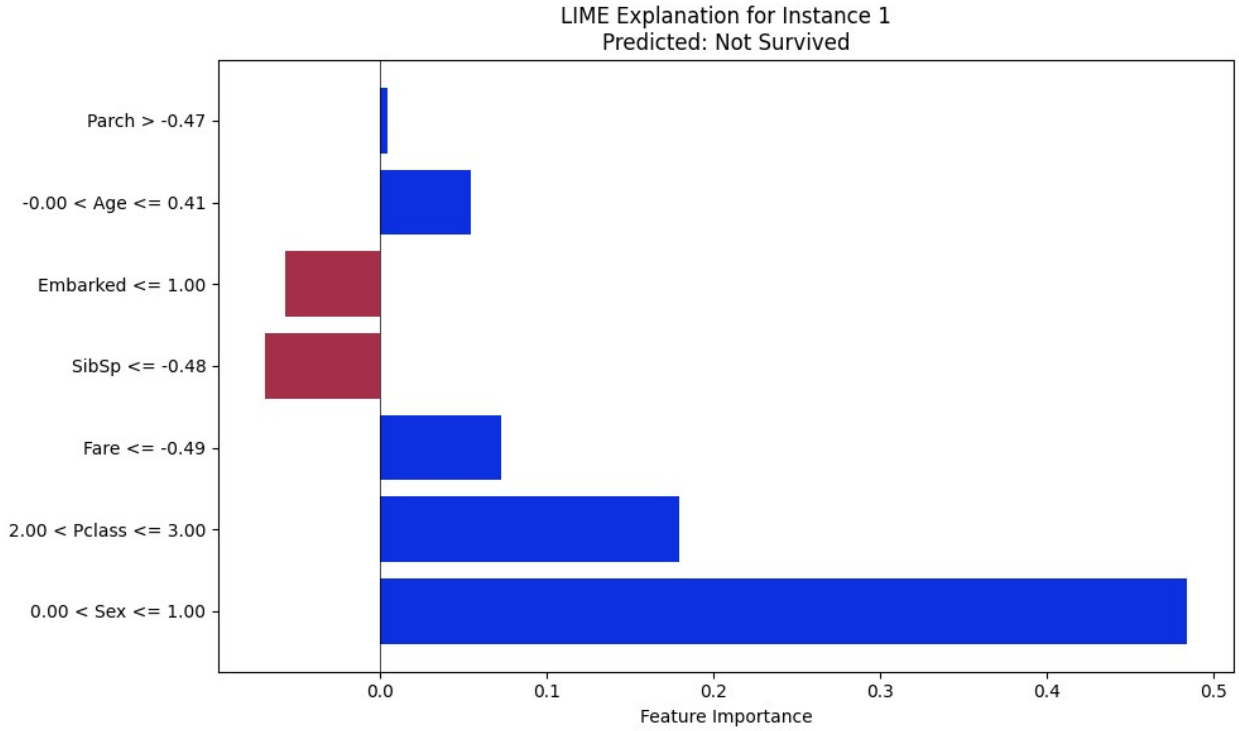
```
# Modified prediction function to return probabilities in the format
LIME expects
def predict_fn(x):
    predictions = titanic_model.predict(x)
    # Ensure we return a 2D array of probabilities for both classes
    return np.array([[1 - p[1], p[1]] for p in predictions])

# Create a LIME explainer for the Titanic dataset
explainer = lime_tabular.LimeTabularExplainer(
    training_data=titanic_train_features.values,
    feature_names=list(titanic_train_features.columns),
    class_names=['Not Survived', 'Survived'],
    mode='classification',
    discretize_continuous=True
)

# Select a few instances to explain
num_explanations = 3
instances_to_explain = titanic_test_features.values[:num_explanations]

# Generate and visualize LIME explanations
for i, instance in enumerate(instances_to_explain):
    explanation = explainer.explain_instance(
        instance,
        predict_fn,
        num_features=len(titanic_train_features.columns),
        top_labels=2
    )

    # Get the predicted class
```

```
    pred_probs = predict_fn(instance.reshape(1, -1))[0]
    pred_class = np.argmax(pred_probs)

    print(f"\nExplanation for Instance {i+1}")
    print(f"Predicted Class: {'Survived' if pred_class == 1 else 'Not
Survived'}")
    print(f"Prediction Probability: {pred_probs[pred_class]:.3f}")

    print("\nFeature Importances:")
    exp_list = explanation.as_list(label=int(pred_class))

    plt.figure(figsize=(10, 6))

    features, weights = zip(*exp_list)

    y_pos = np.arange(len(features))
    plt.barh(y_pos, weights)

    plt.yticks(y_pos, features)
    plt.xlabel('Feature Importance')
    plt.title(f'LIME Explanation for Instance {i+1}\nPredicted:
{"Survived" if pred_class == 1 else "Not Survived"}')

    plt.axvline(x=0, color='black', linestyle='-', linewidth=0.5)

    for j, weight in enumerate(weights):
        if weight > 0:
            plt.barh(j, weight, color='blue', alpha=0.6)
        else:
            plt.barh(j, weight, color='red', alpha=0.6)

    plt.tight_layout()
    plt.show()

    print("\nNumerical Feature Importances:")
    for feat, imp in exp_list:
        print(f"{feat}: {imp:.4f}")
    print("=" * 50)

# Function to compare explanations for different outcomes
def compare_explanations():
    # Find examples from test set instead of training set
    survived_idx = None
    not_survived_idx = None

    predictions = predict_fn(titanic_test_features.values)
    for idx, pred in enumerate(predictions):
        if np.argmax(pred) == 1 and survived_idx is None:
            survived_idx = idx
        elif np.argmax(pred) == 0 and not_survived_idx is None:
```

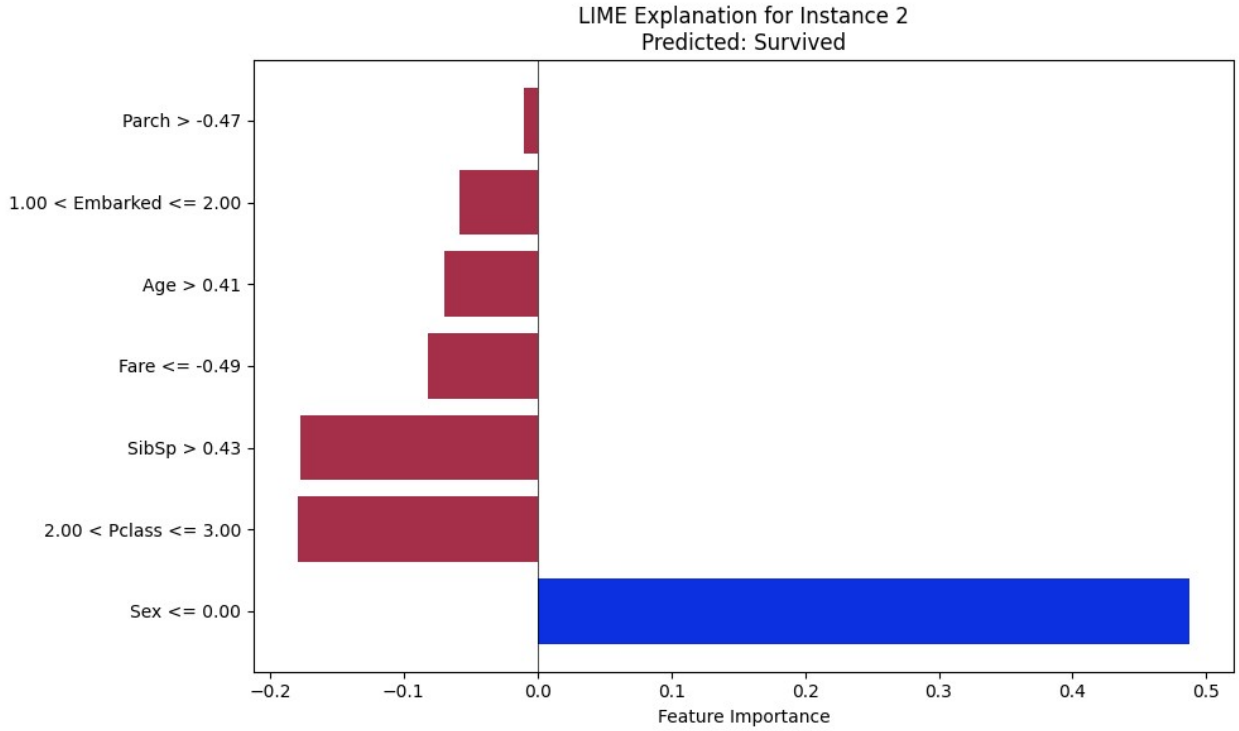
```
            not_survived_idx = idx
        if survived_idx is not None and not_survived_idx is not None:
            break

    # Get explanations for test set instances
    survived_exp = explainer.explain_instance(
        titanic_test_features.values[survived_idx],
        predict_fn,
        num_features=len(titanic_test_features.columns),
        top_labels=2
    )

    not_survived_exp = explainer.explain_instance(
        titanic_test_features.values[not_survived_idx],
        predict_fn,
        num_features=len(titanic_test_features.columns),
        top_labels=2
    )


    # Create comparison plot
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    # Plot for survived prediction
    exp_list_survived = survived_exp.as_list(label=1)
    features_survived, weights_survived = zip(*exp_list_survived)
    y_pos = np.arange(len(features_survived))

    ax1.barh(y_pos, weights_survived)
    ax1.set_yticks(y_pos)
    ax1.set_yticklabels(features_survived)
    ax1.set_xlabel('Feature Importance')
    ax1.set_title('Explanation for Survived Prediction')
    ax1.axvline(x=0, color='black', linestyle='-', linewidth=0.5)

    for j, weight in enumerate(weights_survived):
        if weight > 0:
            ax1.barh(j, weight, color='blue', alpha=0.6)
        else:
            ax1.barh(j, weight, color='red', alpha=0.6)

    # Plot for not survived prediction
    exp_list_not_survived = not_survived_exp.as_list(label=0)
    features_not_survived, weights_not_survived =
zip(*exp_list_not_survived)

    ax2.barh(y_pos, weights_not_survived)
    ax2.set_yticks(y_pos)
    ax2.set_yticklabels(features_not_survived)
    ax2.set_xlabel('Feature Importance')
```

```
    ax2.set_title('Explanation for Not Survived Prediction')
    ax2.axvline(x=0, color='black', linestyle='-', linewidth=0.5)

    for j, weight in enumerate(weights_not_survived):
        if weight > 0:
            ax2.barh(j, weight, color='blue', alpha=0.6)
        else:
            ax2.barh(j, weight, color='red', alpha=0.6)

    plt.tight_layout()
    plt.show()

    # Print detailed comparisons
    print("\nFeature Importances for Survived Case:")
    for feat, imp in exp_list_survived:
        print(f"{feat}: {imp:.4f}")

    print("\nFeature Importances for Not Survived Case:")
    for feat, imp in exp_list_not_survived:
        print(f"{feat}: {imp:.4f}")

# Show comparison of explanations
print("\nComparing explanations between different predictions:")
compare_explanations()
```

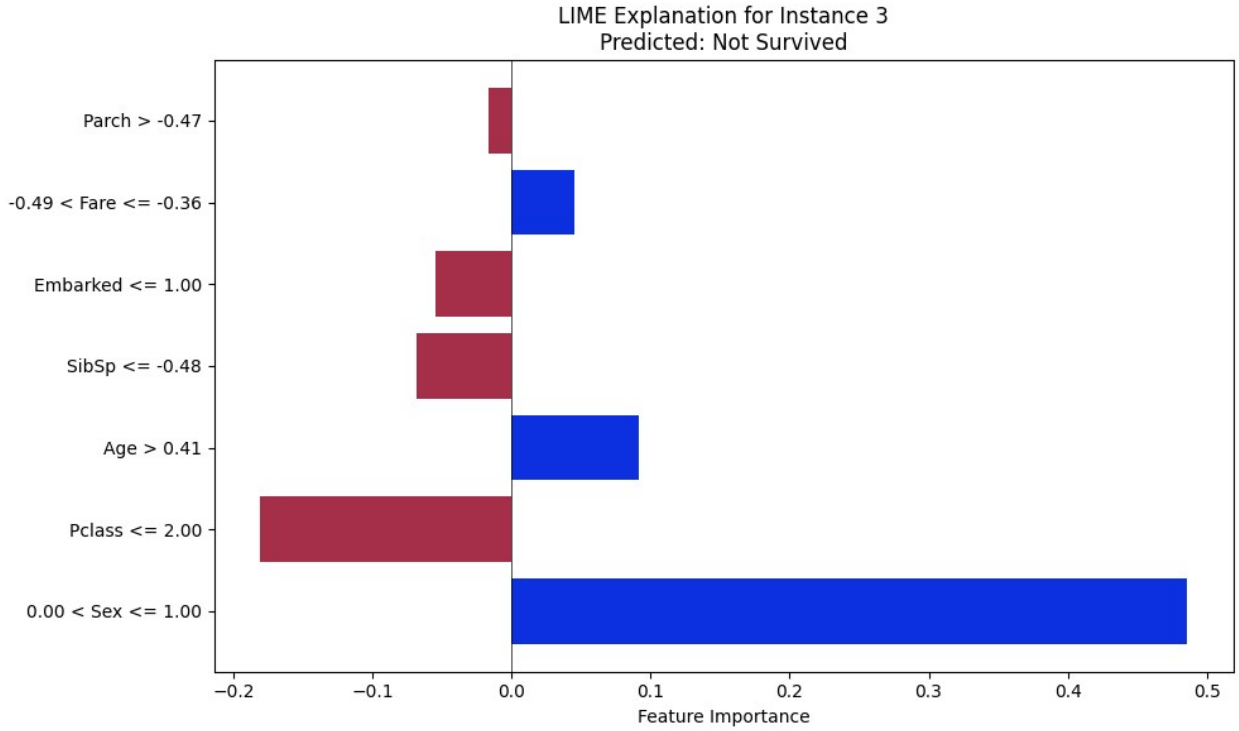
```
Explanation for Instance 1
Predicted Class: Not Survived
Prediction Probability: 0.929

Feature Importances:
```



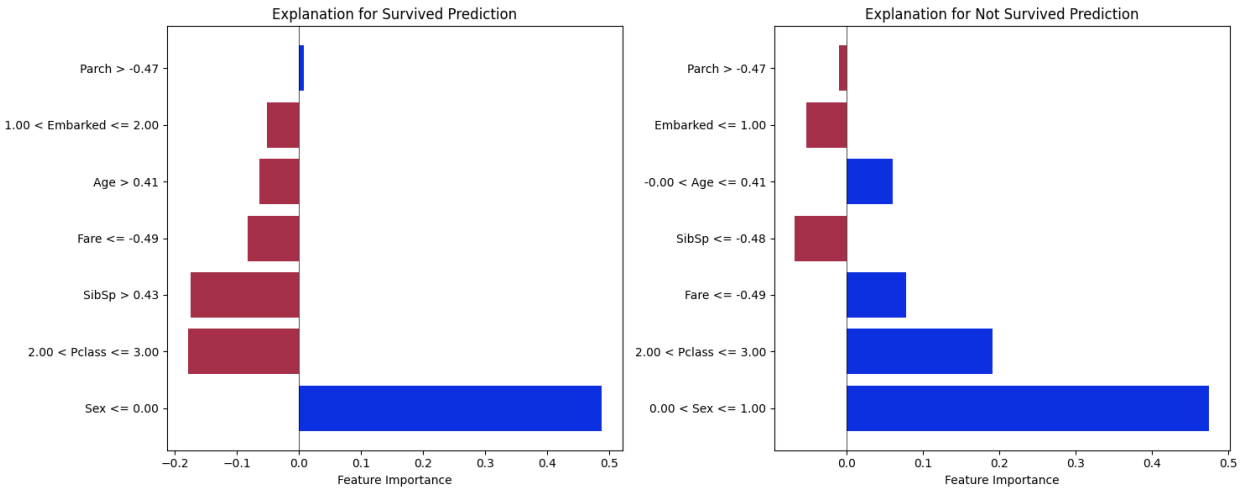

```
Numerical Feature Importances:
0.00 < Sex <= 1.00: 0.4841
2.00 < Pclass <= 3.00: 0.1796
Fare <= -0.49: 0.0729
SibSp <= -0.48: -0.0690
Embarked <= 1.00: -0.0567
-0.00 < Age <= 0.41: 0.0548
Parch > -0.47: 0.0045
==================================================

Explanation for Instance 2
Predicted Class: Survived
Prediction Probability: 0.584

Feature Importances:
```




```
Numerical Feature Importances:
Sex <= 0.00: 0.4871
2.00 < Pclass <= 3.00: -0.1789
SibSp > 0.43: -0.1768
Fare <= -0.49: -0.0819
Age > 0.41: -0.0693
1.00 < Embarked <= 2.00: -0.0586
Parch > -0.47: -0.0105
==================================================

Explanation for Instance 3
Predicted Class: Not Survived
Prediction Probability: 0.957

Feature Importances:
```




```
Numerical Feature Importances:
0.00 < Sex <= 1.00: 0.4859
Pclass <= 2.00: -0.1804
Age > 0.41: 0.0917
SibSp <= -0.48: -0.0682
Embarked <= 1.00: -0.0549
-0.49 < Fare <= -0.36: 0.0453
Parch > -0.47: -0.0167
==================================================

Comparing explanations between different predictions:
```



```
Feature Importances for Survived Case:
Sex <= 0.00: 0.4884
2.00 < Pclass <= 3.00: -0.1794
SibSp > 0.43: -0.1750
Fare <= -0.49: -0.0831
Age > 0.41: -0.0644
1.00 < Embarked <= 2.00: -0.0512
Parch > -0.47: 0.0080

Feature Importances for Not Survived Case:
0.00 < Sex <= 1.00: 0.4746
2.00 < Pclass <= 3.00: 0.1911
Fare <= -0.49: 0.0776
SibSp <= -0.48: -0.0674
-0.00 < Age <= 0.41: 0.0609
Embarked <= 1.00: -0.0525
Parch > -0.47: -0.0100
```

Application of LIME

LIME was applied to explain individual predictions from the feed-forward neural network on the Titanic dataset for three instances with different predicted classes (Survived and Not Survived). LIME explains the model's decision by creating a local surrogate model for the instance being analyzed. It perturbs the input features around the instance and observes how these perturbations influence the prediction. Based on these observations, it fits a linear model that approximates the complex decision boundary of the neural network locally.

For Instance 1, which was predicted as Not Survived with a probability of 0.929, the most important features were Sex (0.4841) and Pclass (0.1796), both contributing positively to the prediction of "Not Survived." Interestingly, features such as SibSp (-0.0690) and Embarked (-0.0567) had negative contributions, suggesting that the individual's family structure and embarkation point mitigated the risk of "Not Survived" slightly.

In Instance 2, predicted as Survived with a probability of 0.584, Sex (0.4871) again emerged as the dominant feature but with a reversed direction compared to Instance 1. Features like Pclass (-0.1789) and SibSp (-0.1768) negatively impacted the likelihood of survival, reflecting the interplay of class and family size in survival chances.

For Instance 3, also predicted as Not Survived with a probability of 0.957, Sex (0.4859) remained the most influential feature, reinforcing its critical role in the model. Pclass (-0.1804) negatively influenced survival, while Age (0.0917) positively contributed, suggesting that youth slightly improved survival probabilities.

Interpretation of Results and Comparison Across Cases

The LIME explanations for the three instances reveal that Sex is consistently the most important feature, demonstrating the model's reliance on this variable for predicting survival. For "Not Survived" predictions (Instances 1 and 3), features like Pclass and Fare contributed positively, aligning with historical biases where lower-class passengers and those with lower fares were less likely to survive. In contrast, for the "Survived" prediction (Instance 2), these features

contributed negatively, reflecting improved survival odds for higher-class passengers and those paying higher fares.

When comparing the explanations for the Survived case versus the Not Survived cases, the direction and magnitude of feature importance differ significantly. For example, Pclass has a negative impact on survival predictions in Instance 2 but a positive impact on "Not Survived" predictions for Instances 1 and 3. Similarly, SibSp contributes negatively in Instance 2 but shows mixed effects in Instances 1 and 3. These differences highlight the model's ability to account for nuanced interactions and local feature effects in individual predictions.

How LIME Approximates the Local Decision Boundary

LIME works by sampling points around the instance being explained and observing how the model's predictions change for these samples. It then fits a simple linear model (the surrogate) to approximate the complex, non-linear decision boundary of the neural network in the vicinity of the instance. The weights of the surrogate model provide feature importance scores, indicating the influence of each feature on the prediction.

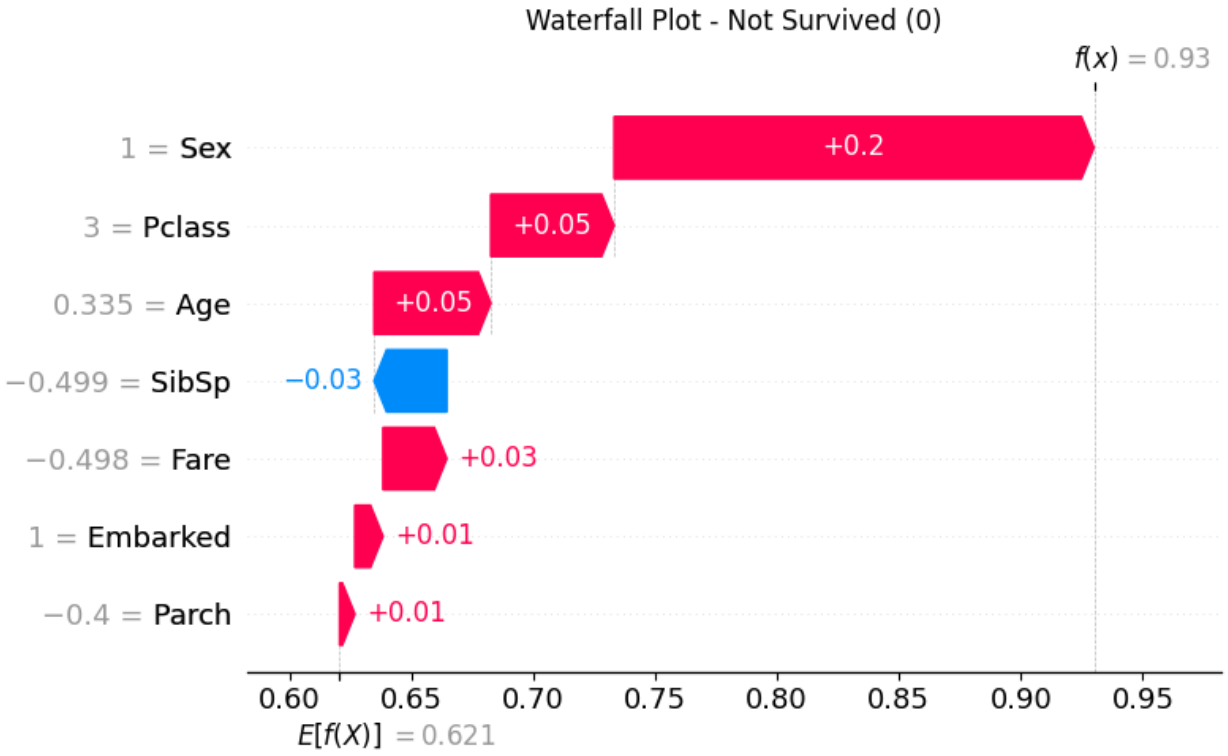
In this analysis, LIME demonstrated its strength in offering interpretable, instance-specific explanations of the neural network's decisions. It highlighted the critical role of Sex, Pclass, and other features while showing how their contributions varied based on local interactions and the specific characteristics of each instance. This makes LIME a valuable tool for gaining insights into the behavior of complex, opaque models at a granular level.

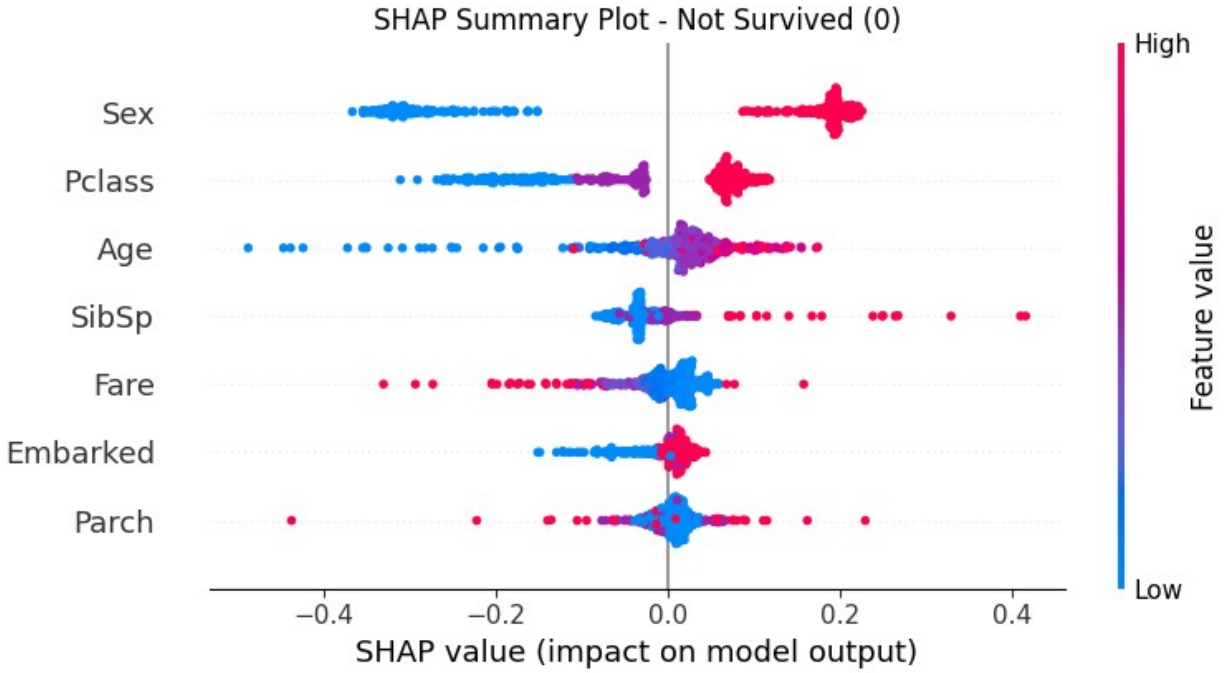
2. Shapley Additive Explanations (SHAP)

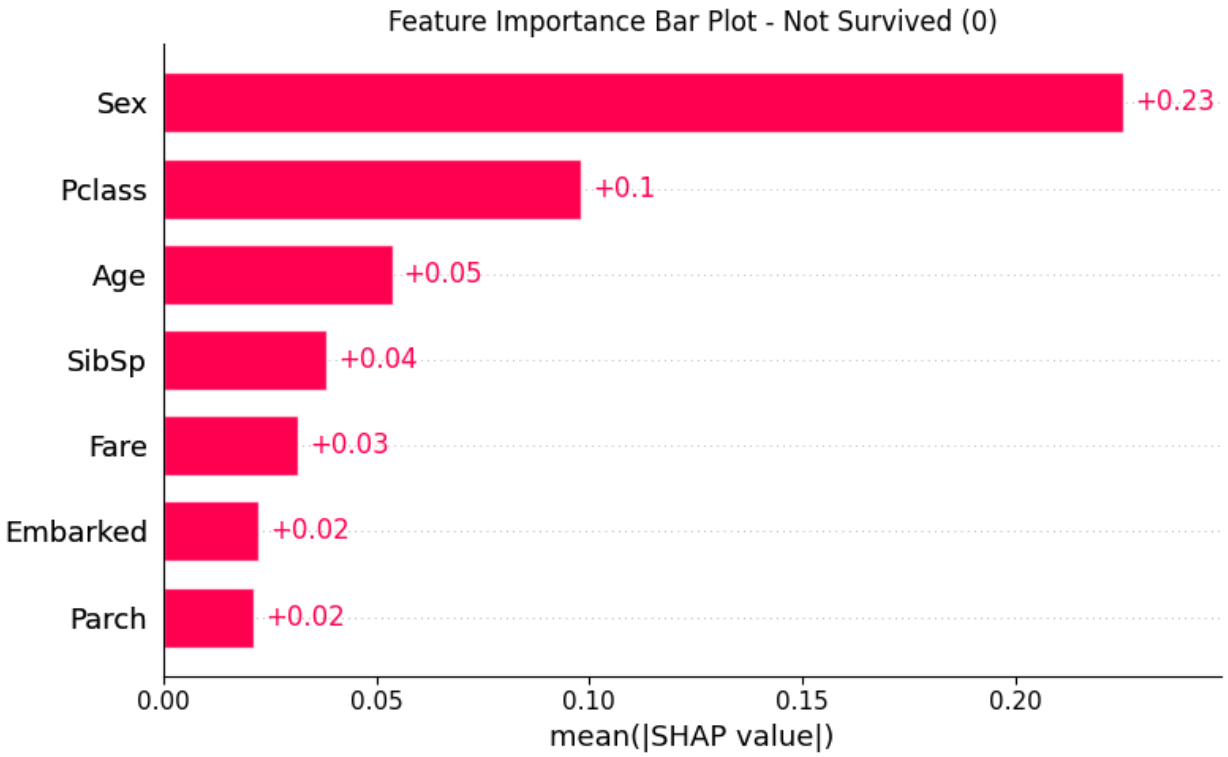
```
# Initialize SHAP explainer for the titanic model
background_data = titanic_train_features.values[:100]  # Use first 100
samples as background
explainer = shap.Explainer(titanic_model, background_data)

# Calculate SHAP values for test set
shap_values = explainer(titanic_test_features.values)

# Feature names mapping
feature_names = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
'Embarked']

# Update the feature names in the shap_values object
shap_values.feature_names = feature_names

# Analyze both classes
class_names = ['Not Survived (0)', 'Survived (1)']
for class_idx, class_name in enumerate(class_names):
    print(f"\n=== SHAP Analysis for {class_name} ===")

    # 1. Waterfall plot for a single prediction
    plt.figure(figsize=(10, 6))
    shap.plots.waterfall(shap_values[0, :, class_idx],
                         show=False,
                         max_display=10)
```

```
    plt.title(f'Waterfall Plot - {class_name}')
    plt.tight_layout()
    plt.show()

    # 2. Summary plot showing feature importance and impact
    plt.figure(figsize=(10, 6))
    shap.summary_plot(shap_values[:, :, class_idx].values,
                      titanic_test_features,
                      feature_names=titanic_test_features.columns,
                      show=False)
    plt.title(f'SHAP Summary Plot - {class_name}')
    plt.tight_layout()
    plt.show()

    # 3. Bar plot of mean absolute SHAP values
    plt.figure(figsize=(10, 6))
    shap.plots.bar(shap_values[:, :, class_idx],
                   show=False)
    plt.title(f'Feature Importance Bar Plot - {class_name}')
    plt.tight_layout()
    plt.show()

    # 4. Feature interaction visualization
    plt.figure(figsize=(10, 6))
    shap.dependence_plot("Age",
                         shap_values[:, :, class_idx].values,
                         titanic_test_features,
                         interaction_index="Fare",
                         show=False)
    plt.title(f'Age-Fare Interaction Plot - {class_name}')
    plt.tight_layout()
    plt.show()


# 5. Additional: Compare feature importance between classes
plt.figure(figsize=(12, 6))
feature_importance = pd.DataFrame({
    'Not Survived': np.abs(shap_values[:, :, 0].values).mean(0),
    'Survived': np.abs(shap_values[:, :, 1].values).mean(0)
}, index=titanic_test_features.columns)

feature_importance.plot(kind='bar')
plt.title('Feature Importance Comparison Between Classes')
plt.xlabel('Features')
plt.ylabel('Mean |SHAP value|')
plt.legend(title='Class')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
ExactExplainer explainer: 419it [00:12, 14.70it/s]

=== SHAP Analysis for Not Survived (0) ===
```

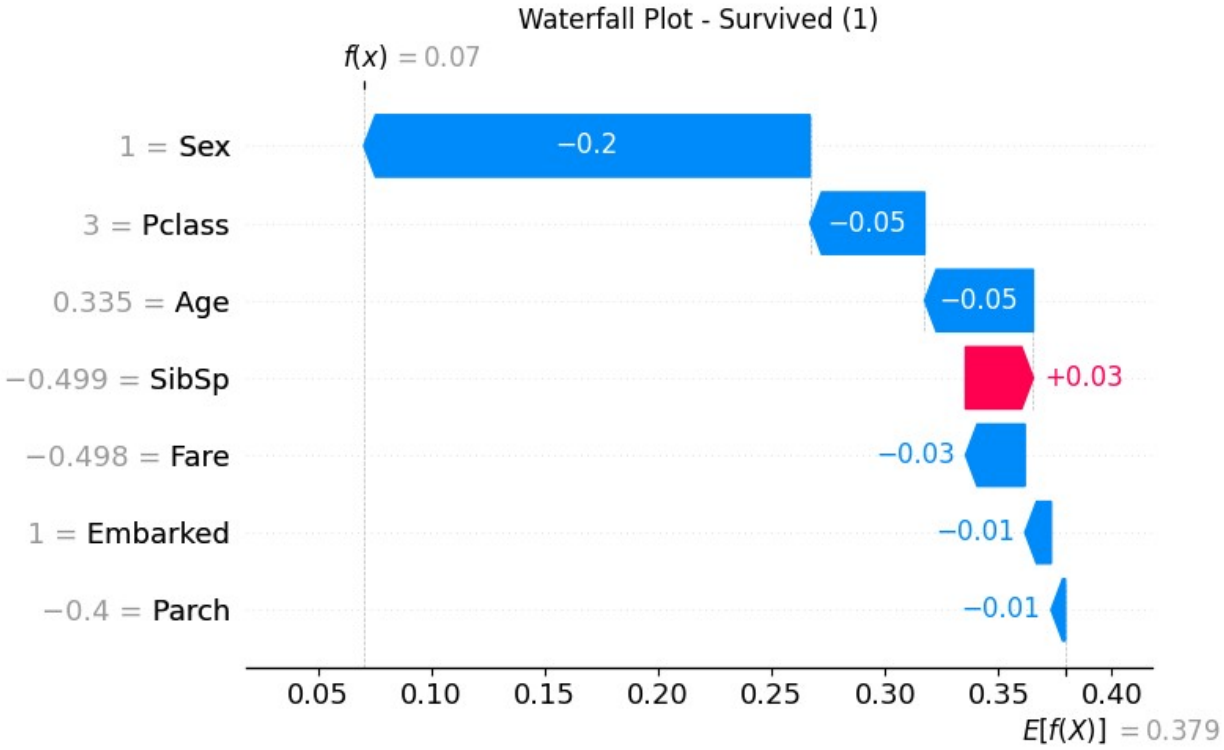


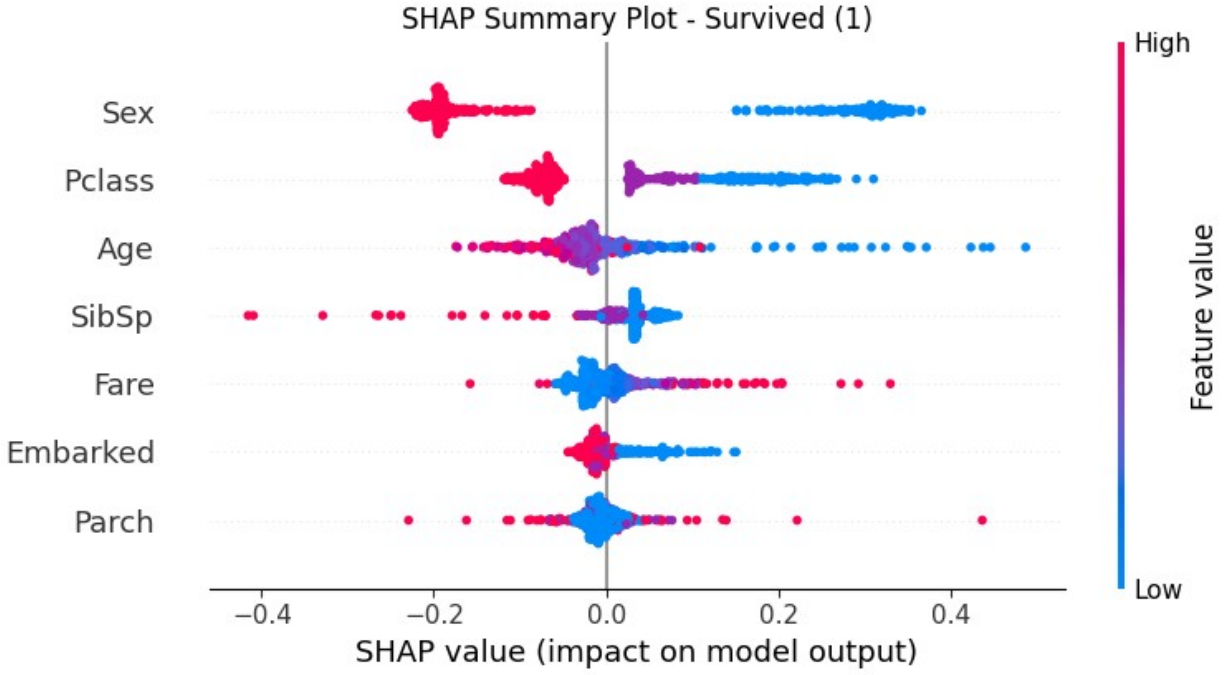


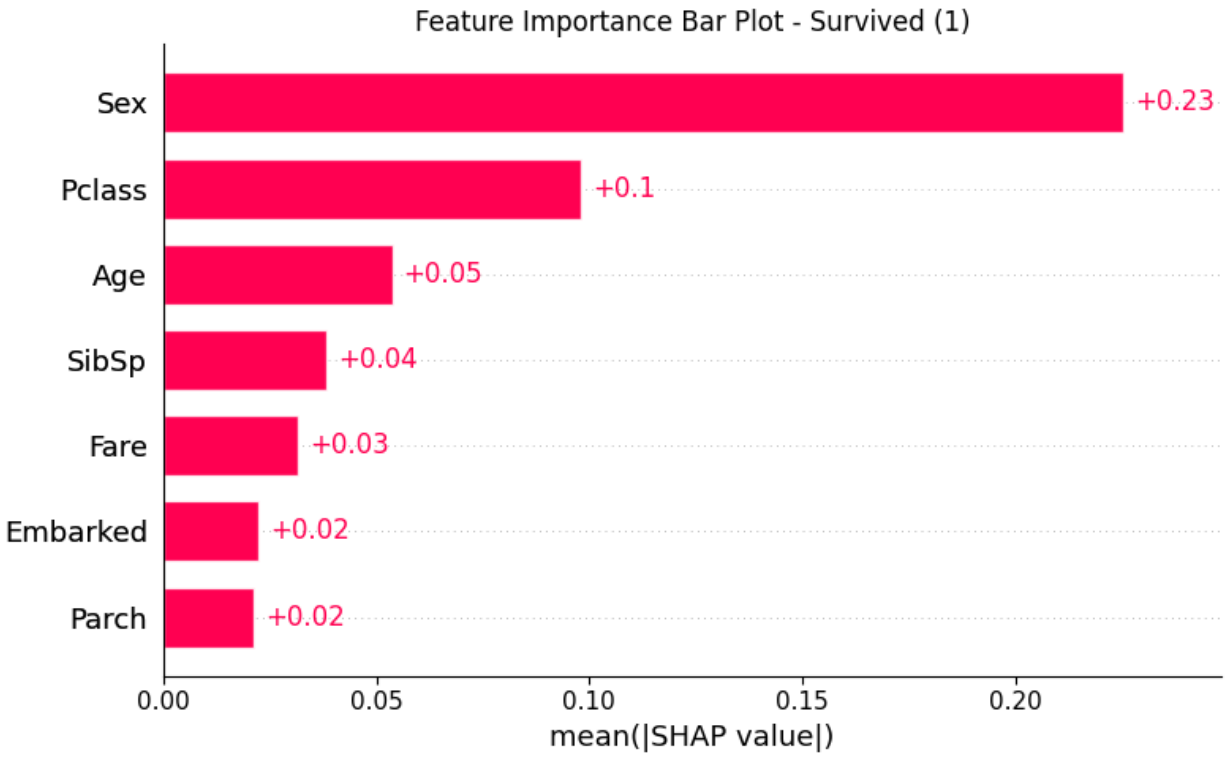



```
<Figure size 1000x600 with 0 Axes>
```


```
=== SHAP Analysis for Survived (1) ===
```

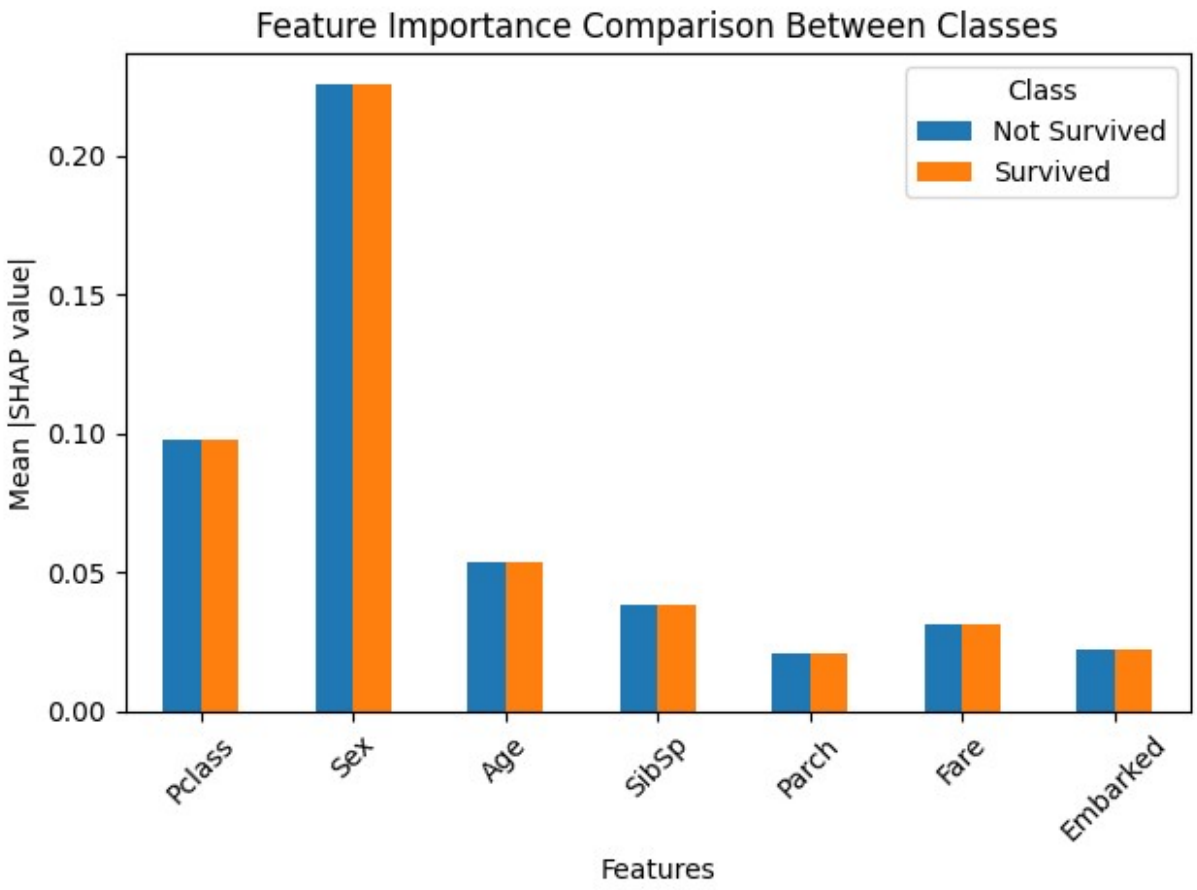

Waterfall Plot - Survived (1)
f(x) = 0.07
1 = Sex
3 = Pclass
0.335 = Age
−0.499 = SibSp
−0.498 = Fare
1 = Embarked
−0.4 = Parch
−0.2
−0.05
−0.05
+0.03
−0.03
−0.01
−0.01
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
E[f(X)] = 0.379


SHAP Summary Plot - Survived (1)
Sex
Pclass
Age
SibSp
Fare
Embarked
Parch
−0.4
−0.2
0.0
0.2
0.4
SHAP value (impact on model output)
High
Feature value
Low




```
<Figure size 1000x600 with 0 Axes>
```


```
<Figure size 1200x600 with 0 Axes>
```



Explanation of Model Predictions Using SHAP

SHAP values provide a comprehensive approach to explaining individual and global model predictions by attributing the contribution of each feature. For the Titanic dataset, the SHAP feature importance analysis confirms that Sex is the most influential predictor for both survival and non-survival cases. For predictions of Survived, the SHAP bar plot and summary plot reveal that Sex contributes significantly (mean SHAP value of ~0.23), followed by Pclass (0.1) and Age (0.05). The waterfall plot for an individual Survived instance illustrates that Sex decreases the log odds of survival, while features like Age and Fare contribute positively. The interaction plot for Age and Fare shows that higher fares increase survival odds, particularly for younger individuals.

For the Not Survived cases, Sex again dominates the feature importance with a similar mean SHAP value of ~0.23, followed by Pclass (0.1) and Age (0.05). The waterfall plot for a specific instance shows that Sex and Pclass heavily impact the prediction toward "Not Survived," while other features like Fare and SibSp contribute less significantly. The interaction plot between Age and Fare for Not Survived cases highlights that low fares and older age contribute significantly to non-survival outcomes.

Comparison Between SHAP and LIME

Both SHAP and LIME are effective for local interpretability, but they differ in methodology and insights:

1. Local Approximation:
    - LIME constructs a local surrogate model for each instance using linear approximations, providing an intuitive view of feature contributions for a specific prediction.
    - SHAP calculates exact contributions based on game-theoretic principles, making it more robust in scenarios with feature interactions.
2. Global Insights:
    - While LIME focuses exclusively on local explanations, SHAP naturally extends to global analysis by aggregating SHAP values across instances. This allows it to explain overall feature importance and interactions.
3. Feature Interaction:
    - SHAP excels in capturing feature interactions, as seen in the interaction plots for Age and Fare, which reveal nuanced relationships between these features. LIME lacks this capability due to its linear surrogate assumption.
4. Consistency and Additivity:
    - SHAP adheres to consistency and additivity principles, ensuring that feature contributions sum to the prediction probability. LIME does not guarantee this, which can result in less consistent explanations.

In summary, while both methods emphasize local interpretability, SHAP provides a more comprehensive analysis due to its robustness, ability to explain global trends, and inclusion of feature interactions. This makes SHAP particularly valuable when exploring both instance-specific predictions and the overall behavior of a model. LIME, on the other hand, is more straightforward to implement and interpret, making it suitable for simpler models or when quick, approximate explanations are sufficient.

3. Anchors

```
def predict_fn(x):
    """Convert model predictions to class labels"""
    predictions = titanic_model.predict(x)
    return np.argmax(predictions, axis=1)

def initialize_anchor_explainer(train_data, feature_names,
categorical_names=None):
    """
    Initialize the AnchorTabular explainer with proper configuration
    """
    categorical_features = [
        i for i, feature in enumerate(feature_names)
        if feature in ['Pclass', 'Sex', 'Embarked']
    ]

    if categorical_names is None:
        categorical_names = {
            1: ['male', 'female'],  # Sex
```

```
            0: [1, 2, 3],          # Pclass
            6: [0, 1, 2]           # Embarked
        }

    explainer = AnchorTabular(
        predictor=predict_fn,
        feature_names=feature_names
    )

    explainer.fit(
        train_data,
        categorical_names=categorical_names,
        categorical_features=categorical_features
    )

    return explainer

def explain_with_anchors(instance, explainer, feature_names,
categorical_names=None):
    """
    Generate and display anchor explanations for a specific instance
    """
    try:
        # Generate explanation
        explanation = explainer.explain(instance)

        # Get prediction
        pred = predict_fn(instance.reshape(1, -1))[0]

        # Print prediction
        print(f"\nPrediction: {'Survived' if pred == 1 else 'Did Not
Survive'}")

        # Print anchor rules
        print("\nAnchor Rules:")
        if explanation.anchor:
            for rule in explanation.anchor:
                print(f"- {rule}")
        else:
            print("No anchor rules found")

        # Print metrics
        print(f"\nPrecision: {explanation.precision:.3f}")
        print(f"Coverage: {explanation.coverage:.3f}")

        # Print mean predictions properly
        if 'mean' in explanation.raw:
            means = explanation.raw['mean']
            if isinstance(means, list):
                print(f"Mean predictions per class: {[f'{m:.3f}' for m
```

```
in means]}")
            else:
                print(f"Mean prediction: {means:.3f}")

        return explanation

    except Exception as e:
        print(f"\nError generating explanation: {str(e)}")
        return None

def analyze_instances(train_data, test_data, feature_names,
n_instances=5):
    """
    Analyze multiple instances using Anchors
    """
    explainer = initialize_anchor_explainer(train_data, feature_names)

    for idx in range(min(n_instances, len(test_data))):
        print(f"\n{'='*50}")
        print(f"Test Instance {idx}:")

        instance = test_data[idx]

        # Print feature values with labels
        print("\nFeature Values:")
        for feature, value in zip(feature_names, instance):
            # Add special formatting for categorical features
            if feature == 'Sex':
                value_str = 'female' if value > 0.5 else 'male'
            elif feature == 'Pclass':
                value_str = f"Class {int(value)}"
            elif feature == 'Embarked':
                ports = {0: 'S', 1: 'C', 2: 'Q'}
                value_str = ports.get(int(value), str(value))
            else:
                value_str = f"{value:.2f}"

            print(f"{feature}: {value_str}")

        explanation = explain_with_anchors(instance, explainer,
feature_names)

# Execute the analysis
feature_names = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
'Embarked']

# Prepare the data
X_train = titanic_train_features.values
X_test = titanic_test_features.values
```

```python
print("Analyzing instances with Anchors...")
analyze_instances(X_train, X_test, feature_names)
```

```
Analyzing instances with Anchors...
1/1 [==============================] - 0s 92ms/step

==================================================
Test Instance 0:

Feature Values:
Pclass: Class 3
Sex: female
Age: 0.33
SibSp: -0.50
Parch: -0.40
Fare: -0.50
Embarked: C
1/1 [==============================] - 0s 22ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
1/1 [==============================] - 0s 21ms/step

Prediction: Did Not Survive

Anchor Rules:
- Sex > 0.00
- Age > -0.59

Precision: 0.964
Coverage: 0.499
Mean predictions per class: ['0.872', '0.964']

==================================================
Test Instance 1:

Feature Values:
Pclass: Class 3
Sex: male
Age: 1.33
SibSp: 0.62
Parch: -0.40
Fare: -0.51
```

```
Embarked: Q
1/1 [==============================] - 0s 23ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
1/1 [==============================] - 0s 23ms/step

Prediction: Did Not Survive

Anchor Rules:
```

```
- SibSp > 0.43
- Fare <= -0.49

Precision: 0.993
Coverage: 0.000
Mean predictions per class: ['0.794', '0.993']

================================================
Test Instance 2:

Feature Values:
Pclass: Class 2
Sex: female
Age: 2.51
SibSp: -0.50
Parch: -0.40
Fare: -0.47
Embarked: C
1/1 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
1/1 [==============================] - 0s 22ms/step

Prediction: Did Not Survive

Anchor Rules:
- Sex > 0.00
- Fare <= -0.36

Precision: 0.978
Coverage: 0.378
Mean predictions per class: ['0.910', '0.978']

================================================
Test Instance 3:

Feature Values:
Pclass: Class 3
Sex: female
Age: -0.26
SibSp: -0.50
Parch: -0.40
Fare: -0.48
Embarked: Q
```

```
1/1 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
1/1 [==============================] - 0s 22ms/step

Prediction: Did Not Survive

Anchor Rules:
- Sex > 0.00
- Age > -0.59

Precision: 0.990
Coverage: 0.503
Mean predictions per class: ['0.940', '0.990']

================================================
Test Instance 4:

Feature Values:
Pclass: Class 3
Sex: male
Age: -0.66
SibSp: 0.62
Parch: 0.62
Fare: -0.42
Embarked: Q
1/1 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
```

```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
1/1 [==============================] - 0s 34ms/step

Prediction: Did Not Survive

Anchor Rules:
- SibSp > 0.43
- Pclass > 2.00

Precision: 0.952
Coverage: 0.065
Mean predictions per class: ['0.787', '0.952']
```

Implementation and Results

Anchors provide interpretable decision rules for individual predictions by identifying a set of feature values that strongly "anchor" a prediction. These rules explain the model's behavior with high precision and coverage. Below are the key results for each test instance:

1. Test Instance 0:
   - Prediction: Did Not Survive
   - Anchor Rules: Sex > 0.00, Age > -0.59
   - Precision: 0.964
   - Coverage: 0.499
   - The anchor rules suggest that being female (Sex > 0.00) and having an age greater than -0.59 heavily influence the prediction of not surviving, with the model consistently predicting the same outcome under these conditions.
2. Test Instance 1:
   - Prediction: Did Not Survive
   - Anchor Rules: SibSp > 0.43, Fare <= -0.49
   - Precision: 0.993
   - Coverage: 0.000
   - The rules indicate that having more siblings or spouses aboard (SibSp > 0.43) and a low fare (Fare <= -0.49) are critical for the prediction. The high precision shows these rules almost always result in the same prediction.
3. Test Instance 2:

- Prediction: Did Not Survive
- Anchor Rules: Sex > 0.00, Fare <= -0.36
- Precision: 0.978
- Coverage: 0.378
- Being female and having a fare less than or equal to -0.36 are the main factors driving the prediction.

4. Test Instance 3:
    - Prediction: Did Not Survive
    - Anchor Rules: Sex > 0.00, Age > -0.59
    - Precision: 0.990
    - Coverage: 0.503
    - The same rules as Instance 0 emerge, reaffirming the significance of gender and age in determining non-survival.
5. Test Instance 4:
    - Prediction: Did Not Survive
    - Anchor Rules: SibSp > 0.43, Pclass > 2.00
    - Precision: 0.952
    - Coverage: 0.065
    - A higher number of siblings/spouses and being in a lower class are influential in predicting non-survival for this instance.

Comparison of Anchors with LIME and SHAP

Anchors differ significantly from LIME and SHAP in their interpretability approach:

1. Local Explanation:
    - Anchors provide if-then rules that describe the minimal conditions required for a prediction with high precision, offering a more logical and rule-based explanation.
    - LIME uses linear surrogates, while SHAP assigns additive contributions to features. These techniques focus on quantifying importance rather than creating rules.
2. Precision and Coverage:
    - Anchors emphasize precision (confidence in prediction accuracy under the anchor conditions) and coverage (fraction of similar instances). This makes Anchors more focused on providing conditions that strongly define a prediction.
    - LIME and SHAP do not explicitly define precision or coverage but instead focus on feature contributions.
3. Simplicity:
    - Anchors create easily understandable rules like "Sex > 0.00," which makes them intuitive for non-technical audiences.
    - SHAP is mathematically rigorous and includes interaction effects but can be less interpretable due to its complexity.

- LIME balances simplicity and accuracy but may oversimplify decision boundaries in complex models.

4. Feature Interaction:
   - SHAP excels in identifying interactions, as seen in the Age-Fare interaction plots, which are not inherently captured by Anchors or LIME.

Summary

Anchors provide interpretable, rule-based explanations that focus on precision and coverage, making them ideal for understanding critical conditions for predictions. However, they lack the depth of SHAP in quantifying feature interactions and the simplicity of LIME for linear explanations. Anchors are particularly useful when users need clear, rule-based decision-making insights.

Part 3: Example-Based Explanations

1. Counterfactual Explanations

Normalize test data and check dimensions

```
X_test = titanic_test_features.values
X_test_norm = (X_test - X_test.min(axis=0)) / (X_test.max(axis=0) -
X_test.min(axis=0))

shape = titanic_train_features.shape[1]

print(f"Expected input dimensions for Counterfactual: {shape}")
print(f"Actual input dimensions being passed: {X_test_norm.shape}")

Expected input dimensions for Counterfactual: 7
Actual input dimensions being passed: (418, 7)
```

Define the counterfactual generator

```
cf = Counterfactual(titanic_model,
                    shape=(1, shape),
                    distance_fn='l1',         # L1 distance for sparse
changes
                    target_proba=0.5,         # target probability for
the desired class
                    max_iter=1000,            # maximum iterations
                    lam_init=1e-1,            # regularization term
                    learning_rate_init=1e-2,  # learning rate
                    feature_range=(0, 1)      # range of feature
values (normalized)
                   )
```

Split data and predict training data

```
X_train = titanic_train_features.values
y_train = np.argmax(titanic_train_labels, axis=1)

train_predictions = titanic_model.predict(X_train)
train_predicted_classes = np.argmax(train_predictions, axis=1)
```

Identify incorrect predictions

```
train_incorrect_indices = np.where(train_predicted_classes != y_train)
[0]

print(f"Number of incorrect predictions in training data:
{len(train_incorrect_indices)}")
print(f"First 10 incorrect prediction indices:
{train_incorrect_indices[:10]}")
```

```
Number of incorrect predictions in training data: 142
First 10 incorrect prediction indices: [14 17 18 21 23 25 36 38 41 49]
```

Select first two and generate counterfactuals

```
selected_train_indices = train_incorrect_indices[:2]

for idx in selected_train_indices:
    instance = X_train[idx].reshape(1, -1)
    explanation = cf.explain(instance)

    print(f"\nIndex: {idx}")
    print("Original Instance (scaled):", instance[0])
    print("Counterfactual Instance (scaled):", explanation.cf['X'][0])
    print(f"Original Prediction: {train_predicted_classes[idx]}
(probability: {train_predictions[idx]
[train_predicted_classes[idx]]:.2f})")
    print(f"Counterfactual Prediction: {np.argmax(explanation.cf['X']
[0])}")
    print("Feature Changes (scaled):", explanation.cf['X'][0] -
instance[0])
    print("=" * 50)
```

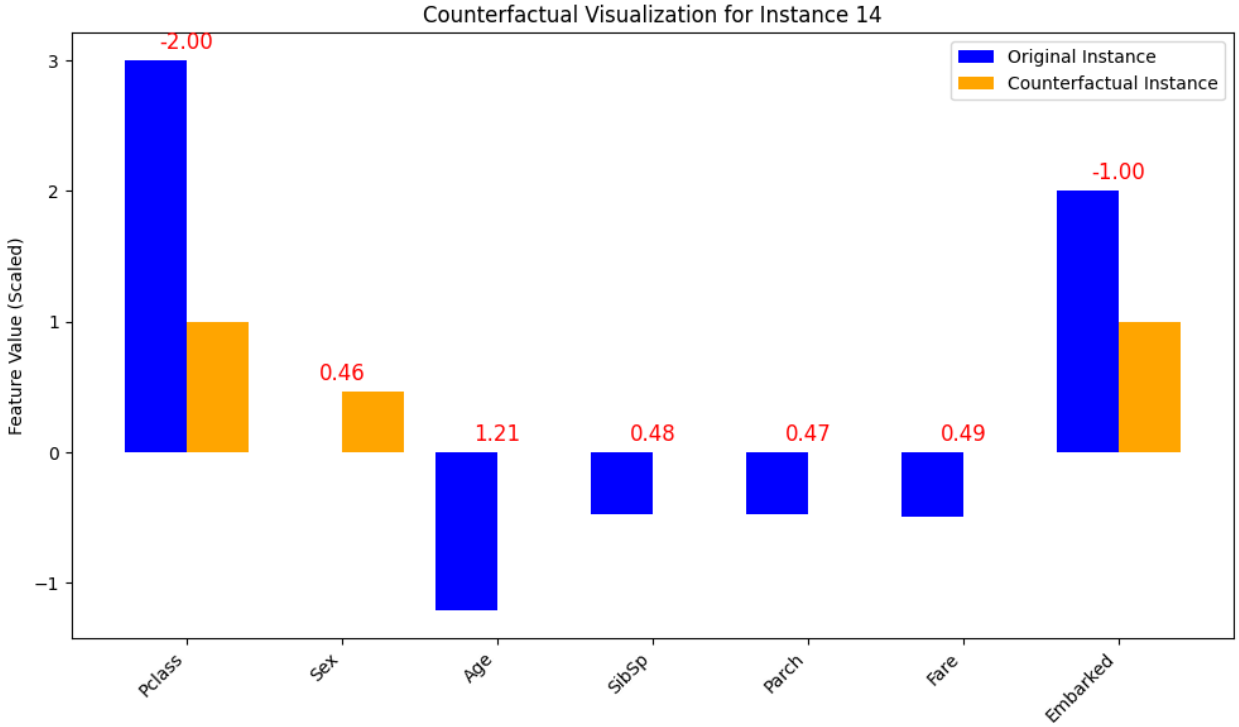
```
Index: 14
Original Instance (scaled): [ 3.          0.         -1.20685378 -
0.47519908 -0.47432585 -0.48807536
  2.        ]
Counterfactual Instance (scaled): [1.         0.46452063 0.         0.
0.         0.
 1.        ]
Original Prediction: 1 (probability: 0.60)
Counterfactual Prediction: 0
Feature Changes (scaled): [-2.          0.46452063  1.20685378
```

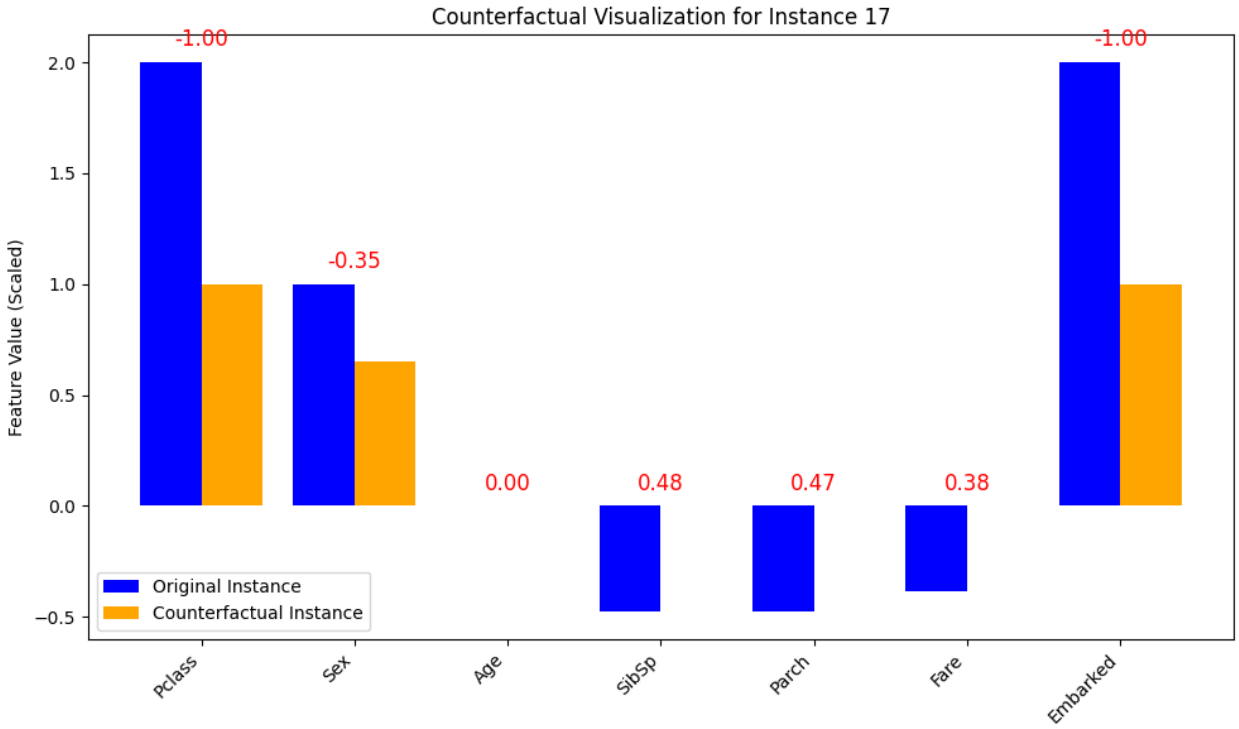
```
0.47519908  0.47432585  0.48807536
 -1.        ]
==================================================

Index: 17
Original Instance (scaled): [ 2.00000000e+00  1.00000000e+00 -
5.48213849e-16 -4.75199081e-01
 -4.74325852e-01 -3.84474648e-01  2.00000000e+00]
Counterfactual Instance (scaled): [1.        0.6528089 0.        0.
0.        0.        1.       ]
Original Prediction: 0 (probability: 0.81)
Counterfactual Prediction: 0
Feature Changes (scaled): [-1.00000000e+00 -3.47191095e-01
5.48213849e-16  4.75199081e-01
  4.74325852e-01  3.84474648e-01 -1.00000000e+00]
==================================================
```

Visualise both instances

```
def visualize_counterfactual(index, original, counterfactual,
features):
    changes = counterfactual - original

    plt.figure(figsize=(10, 6))
    x = np.arange(len(features))

    plt.bar(x - 0.2, original, width=0.4, label='Original Instance',
color='blue')
    plt.bar(x + 0.2, counterfactual, width=0.4, label='Counterfactual
Instance', color='orange')

    for i, change in enumerate(changes):
        if change != 0:
            plt.text(x[i], max(original[i], counterfactual[i]) + 0.05,
f'{change:.2f}',
                     ha='center', va='bottom', fontsize=12,
color='red')

    plt.xticks(x, features, rotation=45, ha='right')
    plt.ylabel('Feature Value (Scaled)')
    plt.title(f'Counterfactual Visualization for Instance {index}')
    plt.legend()
    plt.tight_layout()
    plt.show()

feature_names = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
'Embarked']

for idx in selected_train_indices:
    original = X_train[idx]
```

```
    counterfactual = cf.explain(original.reshape(1, -1)).cf['X'][0]
    visualize_counterfactual(idx, original, counterfactual,
feature_names)
```





2. Prototypes and Criticisms

Convert to list of lists

```
X_list = titanic_train_data.values.tolist()
print(f"Preprocessed Titanic data shape: {len(X_list)} samples with
{len(X_list[0])} features")
```

```
Preprocessed Titanic data shape: 889 samples with 8 features
```

Initialise MMDCritic

```
critic = MMDCritic(X_list, RBFKernel(1), RBFKernel(0.025))
```

Select 5 prototypes and 10 criticisms

```
protos, _ = critic.select_prototypes(5)

print(f"Selected 5 prototypes:")
for i, proto in enumerate(protos):
    print(f"Prototype {i + 1}: {proto}")
```

```
Selected 5 prototypes:
Prototype 1: [ 0.          3.          1.         -0.12669441 -
0.47519908 -0.47432585
 -0.45494036  2.        ]
Prototype 2: [ 1.          1.          1.          0.49053951 -
0.47519908 -0.47432585
 -0.11494329  2.        ]
Prototype 3: [ 1.          3.          0.         -0.58961986 -
0.47519908 -0.47432585
 -0.49017322  1.        ]
Prototype 4: [ 0.          3.          1.          1.64785311 -
0.47519908 -0.47432585
 -0.4841333   2.        ]
Prototype 5: [ 0.          2.          1.         -0.6667741
0.43135024 -0.47432585
 -0.41467424  2.        ]
```

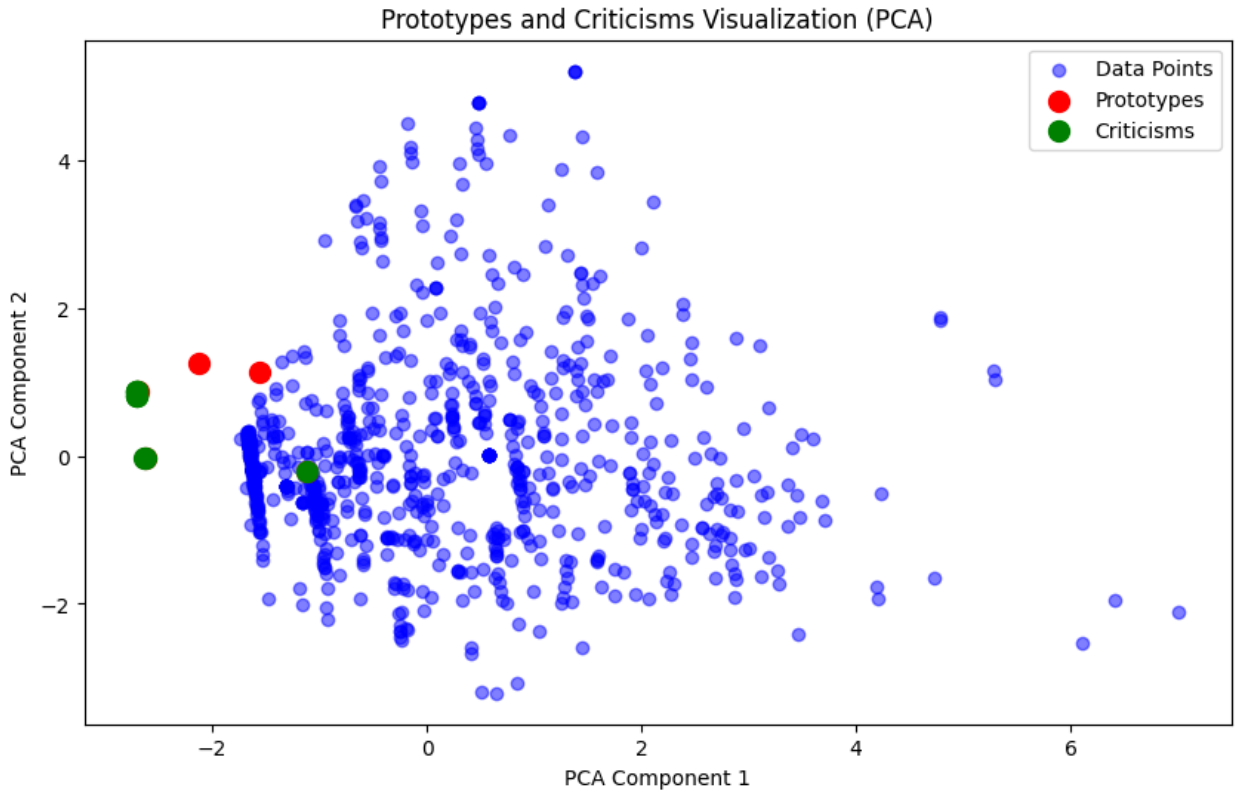
```
criticisms, _ = critic.select_criticisms(10, protos)

print(f"Selected 10 criticisms:")
for i, crit in enumerate(criticisms):
    print(f"Criticism {i + 1}: {crit}")
```

```
Selected 10 criticisms:
Criticism 1: [ 1.          1.          1.          0.49053951 -
0.47519908 -0.47432585
 -0.1169566   2.        ]
Criticism 2: [ 0.          3.          1.          1.64785311 -
```

```
0.47519908 -0.47432585
 -0.49017322  2.        ]
Criticism 3: [ 0.          3.          1.         -0.12669441 -
0.47519908 -0.47432585
 -0.48723782  2.        ]
Criticism 4: [ 0.          3.          1.         -0.12669441 -
0.47519908 -0.47432585
 -0.48723782  2.        ]
Criticism 5: [ 0.          3.          1.         -0.12669441 -
0.47519908 -0.47432585
 -0.48807536  2.        ]
Criticism 6: [ 0.          3.          1.         -0.12669441 -
0.47519908 -0.47432585
 -0.48925113  2.        ]
Criticism 7: [ 0.          3.          1.          1.64785311 -
0.47519908 -0.47432585
 -0.5041818   2.        ]
Criticism 8: [ 0.00000000e+00  3.00000000e+00  1.00000000e+00 -
5.48213849e-16
 -4.75199081e-01 -4.74325852e-01 -4.87237820e-01  2.00000000e+00]
Criticism 9: [ 0.00000000e+00  3.00000000e+00  1.00000000e+00 -
5.48213849e-16
 -4.75199081e-01 -4.74325852e-01 -4.87237820e-01  2.00000000e+00]
Criticism 10: [ 0.00000000e+00  3.00000000e+00  1.00000000e+00 -
5.48213849e-16
 -4.75199081e-01 -4.74325852e-01 -4.87237820e-01  2.00000000e+00]
```

Visualise prototypes and criticism

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(np.array(X_list))

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
protos_pca = pca.transform(np.array(protos))
criticisms_pca = pca.transform(np.array(criticisms))

plt.figure(figsize=(10, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c='b', alpha=0.5, label='Data
Points')
plt.scatter(protos_pca[:, 0], protos_pca[:, 1], c='r',
label='Prototypes', s=100)
plt.scatter(criticisms_pca[:, 0], criticisms_pca[:, 1], c='g',
label='Criticisms', s=100)
plt.title('Prototypes and Criticisms Visualization (PCA)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()
```



The visualisation of prototypes and comments allows for a clear differentiation between representative and challenging data points in the dataset. The prototypes, indicated in red, are typical cases that closely match the model's learnt decision boundary, resulting in precise and confident predictions. For example, the PCA projection shows that these prototypes are centrally positioned inside dense clusters of data. In contrast, the criticisms, highlighted in green, correspond to data points that are more dispersed and depart significantly from the main clusters, implying that they challenge the model's generalisation ability. The highlighted criticisms, such as instances with atypical feature combinations (for example, higher "Pclass" with lower "Age" or strange fare values), are most likely due to edge cases or noise in the dataset. By analysing these criticisms, potential data quality issues or model flaws can be identified. This process of discovering prototypes and criticisms improves the model's interpretability by revealing where it operates consistently and where it suffers, allowing for improvements in debugging and development of the model.

==========
==========