Data Science with Python - Awesome Tutorials and Fuel Efficiency Prediction with Regression using The Auto MPG dataset

```
This kernel contains Prediction of Fuel Efficiency - Regression using The Auto MPG
dataset and a curated list of Python tutorials for Data Science, NLP and Machine Le
arning.
```

> **Credits: Thanks to TensorFlow Team, Ujjwal Karn and other contributers for such wonderful curated collections!**

Here are some of *my kernel notebooks* for Machine Learning and Data Science as follows, *Upvote* them if you *like* them

> - Awesome Deep Learning Basics and Resources
> - Data Science with R - Awesome Tutorials
> - Data Science and Machine Learning Cheetcheets
> - Awesome ML Frameworks and MNIST Classification
> - Awesome Data Science for Beginners with Titanic Exploration
> - Tensorflow Tutorial and House Price Prediction
> - Practical Machine Learning with PyTorch
> - Data Scientist's Toolkits - Awesome Data Science Resources
> - Awesome Computer Vision Resources (TBU)
> - Machine Learning Engineer's Toolkit with Roadmap
> - Awesome TensorFlow and PyTorch Resources
> - Hands-on ML with scikit-learn and TensorFlow
> - Awesome Data Science IPython Notebooks
> - Machine Learning and Deep Learning - Awesome Tutorials
> - Data Science with Python - Awesome Tutorials

Fuel Efficiency Prediction - Regression using The Auto MPG dataset

In a *regression* problem, we aim to predict the output of a continuous value, like a price or a probability. Contrast this with a *classification* problem, where we aim to select a class from a list of classes (for example, where a picture contains an apple or an orange, recognizing which fruit is in the picture).

This notebook uses the classic Auto MPG Dataset and builds a model to predict the fuel efficiency of late-1970s and early 1980s automobiles. To do this, we'll provide the model with a description of many automobiles from that time period. This description includes attributes like: cylinders, displacement, horsepower, and weight.

This example uses the `tf.keras` API, see this guide for details.

In [1]:

```
# Use seaborn for pairplot
!pip install seaborn
```

```
Requirement already satisfied: seaborn in /opt/conda/lib/python3.6/site-packages (0.9.0)
Requirement already satisfied: pandas>=0.15.2 in /opt/conda/lib/python3.6/site-packages (
from seaborn) (0.23.4)
Requirement already satisfied: matplotlib>=1.4.3 in /opt/conda/lib/python3.6/site-package
s (from seaborn) (2.2.3)
Requirement already satisfied: numpy>=1.9.3 in /opt/conda/lib/python3.6/site-packages (fr
om seaborn) (1.16.1)
om seaborn) (1.10.1)
Requirement already satisfied: scipy>=0.14.0 in /opt/conda/lib/python3.6/site-packages (f
rom seaborn) (1.1.0)
Requirement already satisfied: python-dateutil>=2.5.0 in /opt/conda/lib/python3.6/site-pa
ckages (from pandas>=0.15.2->seaborn) (2.6.0)
Requirement already satisfied: pytz>=2011k in /opt/conda/lib/python3.6/site-packages (fro
m pandas>=0.15.2->seaborn) (2018.4)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.6/site-package
s (from matplotlib>=1.4.3->seaborn) (1.0.1)
Requirement already satisfied: six>=1.10 in /opt/conda/lib/python3.6/site-packages (from
matplotlib>=1.4.3->seaborn) (1.12.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.6/site-packages (fr
om matplotlib>=1.4.3->seaborn) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /opt/conda/lib
/python3.6/site-packages (from matplotlib>=1.4.3->seaborn) (2.2.0)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.6/site-packages (from
kiwisolver>=1.0.1->matplotlib>=1.4.3->seaborn) (39.1.0)
You are using pip version 18.1, however version 19.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

In [2]:

```
from __future__ import absolute_import, division, print_function

import pathlib

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)
```

```
1.12.0
```

The Auto MPG dataset

The dataset is available from the UCI Machine Learning Repository.

Get the data

First download the dataset.

In [3]:

```
dataset_path = keras.utils.get_file("auto-mpg.data", "https://archive.ics.uci.edu/ml/mach
ine-learning-databases/auto-mpg/auto-mpg.data")
dataset_path
```

```
Downloading data from https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/
auto-mpg.data
32768/30286 [=================================] - 0s 4us/step
```

Out[3]:

```
'/tmp/.keras/datasets/auto-mpg.data'
```

Import it using pandas

In [4]:

```
column_names = ['MPG','Cylinders','Displacement','Horsepower','Weight',
                'Acceleration', 'Model Year', 'Origin']
raw_dataset = pd.read_csv(dataset_path, names=column_names,
                      na_values = "?", comment='\t',
                      sep=" ", skipinitialspace=True)
```

```
dataset = raw_dataset.copy()
dataset.tail()
```

Out[4]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
393	27.0	4	140.0	86.0	2790.0	15.6	82	1
394	44.0	4	97.0	52.0	2130.0	24.6	82	2
395	32.0	4	135.0	84.0	2295.0	11.6	82	1
396	28.0	4	120.0	79.0	2625.0	18.6	82	1
397	31.0	4	119.0	82.0	2720.0	19.4	82	1

Clean the data

The dataset contains a few unknown values.

In [5]:

```
dataset.isna().sum()
```

Out[5]:

```
MPG             0
Cylinders       0
Displacement    0
Horsepower      6
Weight          0
Acceleration    0
Model Year      0
Origin          0
dtype: int64
```

To keep this initial tutorial simple drop those rows.

In [6]:

```
dataset = dataset.dropna()
```

The `"Origin"` column is really categorical, not numeric. So convert that to a one-hot:

In [7]:

```
origin = dataset.pop('Origin')
```

In [8]:

```
dataset['USA'] = (origin == 1)*1.0
dataset['Europe'] = (origin == 2)*1.0
dataset['Japan'] = (origin == 3)*1.0
dataset.tail()
```

Out[8]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	USA	Europe	Japan
393	27.0	4	140.0	86.0	2790.0	15.6	82	1.0	0.0	0.0
394	44.0	4	97.0	52.0	2130.0	24.6	82	0.0	1.0	0.0
395	32.0	4	135.0	84.0	2295.0	11.6	82	1.0	0.0	0.0
396	28.0	4	120.0	79.0	2625.0	18.6	82	1.0	0.0	0.0
397	31.0	4	119.0	82.0	2720.0	19.4	82	1.0	0.0	0.0

Split the data into train and test

Now split the dataset into a training set and a test set.

We will use the test set in the final evaluation of our model.

In [9]:

```
train_dataset = dataset.sample(frac=0.8,random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

Inspect the data

Have a quick look at the joint distribution of a few pairs of columns from the training set.

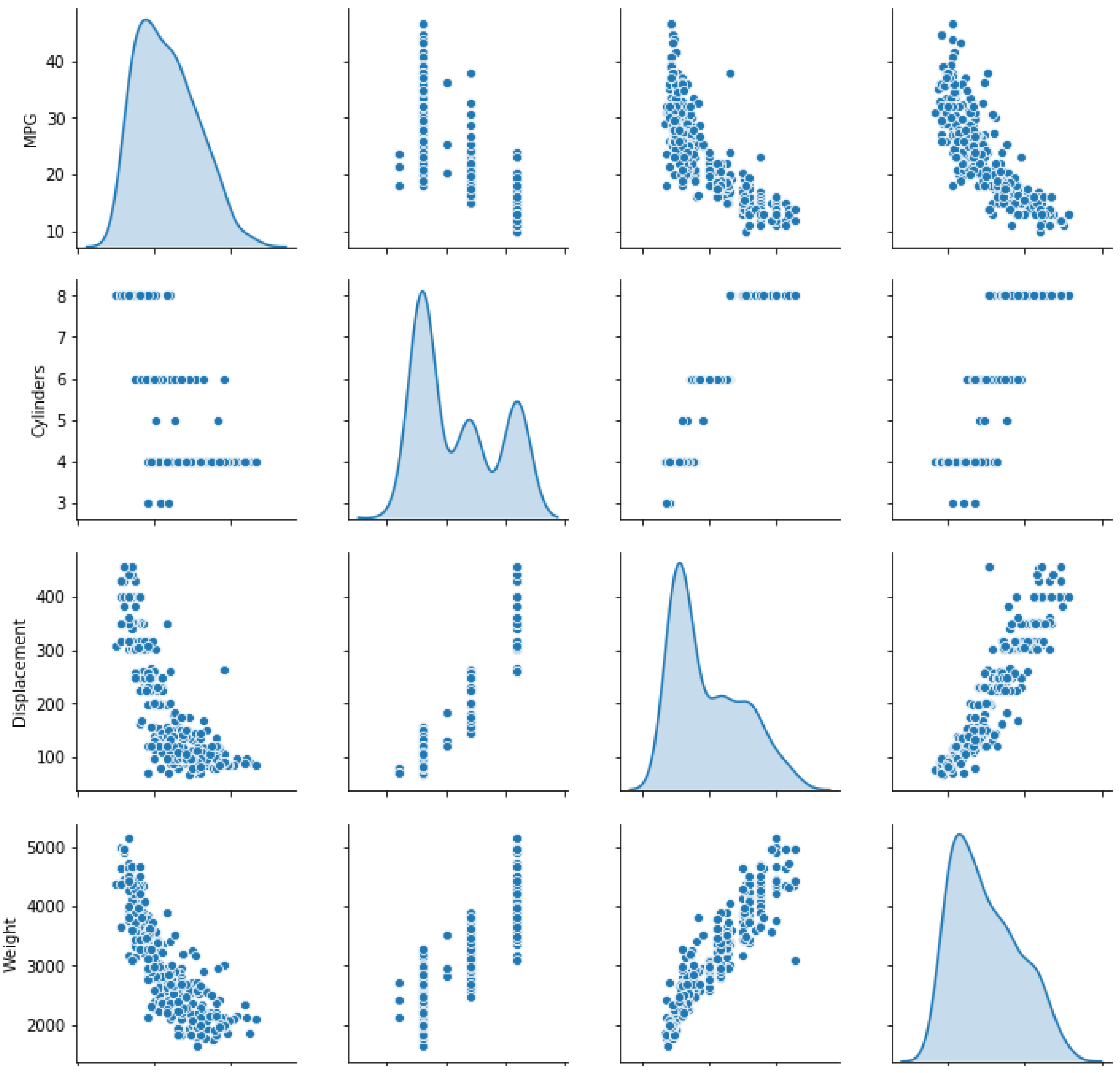
In [10]:

```
sns.pairplot(train_dataset[["MPG", "Cylinders", "Displacement", "Weight"]], diag_kind="kd
e")
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a
non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` ins
tead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.arr
ay(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x7f750c1a23c8>
```



Also look at the overall statistics:

In [11]:

```
train_stats = train_dataset.describe()
train_stats.pop("MPG")
train_stats = train_stats.transpose()
train_stats
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
Cylinders	314.0	5.477707	1.699788	3.0	4.00	4.0	8.00	8.0
Displacement	314.0	195.318471	104.331589	68.0	105.50	151.0	265.75	455.0
Horsepower	314.0	104.869427	38.096214	46.0	76.25	94.5	128.00	225.0
Weight	314.0	2990.251592	843.898596	1649.0	2256.50	2822.5	3608.00	5140.0
Acceleration	314.0	15.559236	2.789230	8.0	13.80	15.5	17.20	24.8
Model Year	314.0	75.898089	3.675642	70.0	73.00	76.0	79.00	82.0
USA	314.0	0.624204	0.485101	0.0	0.00	1.0	1.00	1.0
Europe	314.0	0.178344	0.383413	0.0	0.00	0.0	0.00	1.0
Japan	314.0	0.197452	0.398712	0.0	0.00	0.0	0.00	1.0

Split features from labels

Separate the target value, or "label", from the features. This label is the value that you will train the model to predict.

In [12]:

```
train_labels = train_dataset.pop('MPG')
test_labels = test_dataset.pop('MPG')
```

Normalize the data

Look again at the `train_stats` block above and note how different the ranges of each feature are.

It is good practice to normalize features that use different scales and ranges. Although the model *might* converge without feature normalization, it makes training more difficult, and it makes the resulting model dependent on the choice of units used in the input.

Note: Although we intentionally generate these statistics from only the training dataset, these statistics will also be used to normalize the test dataset. We need to do that to project the test dataset into the same distribution that the model has been trained on.

In [13]:

```
def norm(x):
  return (x - train_stats['mean']) / train_stats['std']
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)
```

This normalized data is what we will use to train the model.

Caution: The statistics used to normalize the inputs here (mean and standard deviation) need to be applied to any other data that is fed to the model, along with the one-hot encoding that we did earlier. That includes the test set as well as live data when the model is used in production.

The model

Build the model

Let's build our model. Here, we'll use a `Sequential` model with two densely connected hidden layers, and an output layer that returns a single, continuous value. The model building steps are wrapped in a function, `build_model`, since we'll create a second model, later on.

In [14]:

```
def build_model():
  model = keras.Sequential([
    layers.Dense(64, activation=tf.nn.relu, input_shape=[len(train_dataset.keys())]),
    layers.Dense(64, activation=tf.nn.relu),
    layers.Dense(1)
  ])

  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mean_squared_error',
                optimizer=optimizer,
                metrics=['mean_absolute_error', 'mean_squared_error'])
  return model
```

In [15]:

```
model = build_model()
```

Inspect the model

Use the `.summary` method to print a simple description of the model

In [16]:

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 64)                640       
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65        
=================================================================
Total params: 4,865
Trainable params: 4,865
Non-trainable params: 0
_________________________________________________________________
```

Now try out the model. Take a batch of `10` examples from the training data and call `model.predict` on it.

In [17]:

```
example_batch = normed_train_data[:10]
example_result = model.predict(example_batch)
example_result
```

Out[17]:

```
array([[-0.12670723],
       [-0.03443428],
       [ 0.3062502 ],
       [ 0.3065169 ],
       [ 0.36841604],
       [ 0.02191051],
       [ 0.3674207 ],
       [ 0.10561748],
       [ 0.00638346],
       [-0.00226454]], dtype=float32)
```

It seems to be working, and it produces a result of the expected shape and type.

Train the model

Train the model for 1000 epochs, and record the training and validation accuracy in the `history` object.

In [18]:

```python
# Display training progress by printing a single dot for each completed epoch
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

EPOCHS = 1000

history = model.fit(
  normed_train_data, train_labels,
  epochs=EPOCHS, validation_split = 0.2, verbose=0,
  callbacks=[PrintDot()])
```

```
.........................................................................................
...........
.........................................................................................
...........
.........................................................................................
...........
.........................................................................................
...........
.........................................................................................
...........
.........................................................................................
...........
.........................................................................................
...........
.........................................................................................
...........
.........................................................................................
...........
.........................................................................................
...........
```

Visualize the model's training progress using the stats stored in the `history` object.

In [19]:

```python
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

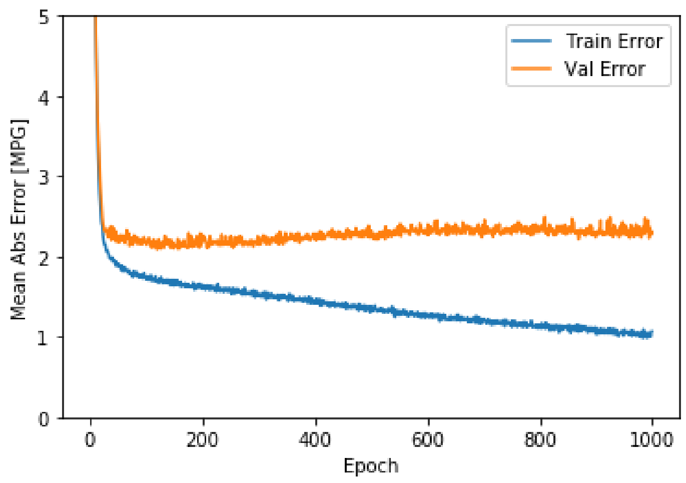
Out[19]:

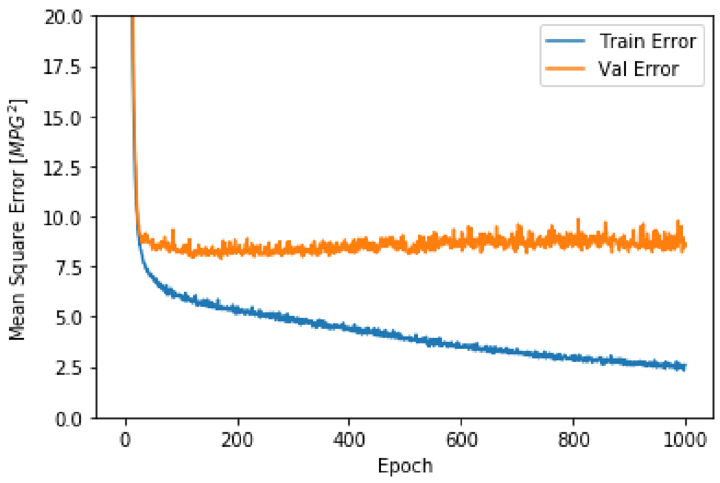
	val_loss	val_mean_absolute_error	val_mean_squared_error	loss	mean_absolute_error	mean_squared_error	epoch
995	8.969163	2.363666	8.969163	2.447986	0.999695	2.447986	995
996	8.801920	2.325327	8.801920	2.558123	1.016450	2.558123	996
997	8.814242	2.317156	8.814242	2.315611	0.999247	2.315611	997
998	8.445328	2.270796	8.445328	2.501316	1.058569	2.501316	998
999	8.631644	2.305249	8.631644	2.599803	1.068485	2.599803	999

In [20]:

In [20]:

```
def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error [MPG]')
  plt.plot(hist['epoch'], hist['mean_absolute_error'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_absolute_error'],
           label = 'Val Error')
  plt.ylim([0,5])
  plt.legend()

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Square Error [$MPG^2$]')
  plt.plot(hist['epoch'], hist['mean_squared_error'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_squared_error'],
           label = 'Val Error')
  plt.ylim([0,20])
  plt.legend()
  plt.show()


plot_history(history)
```





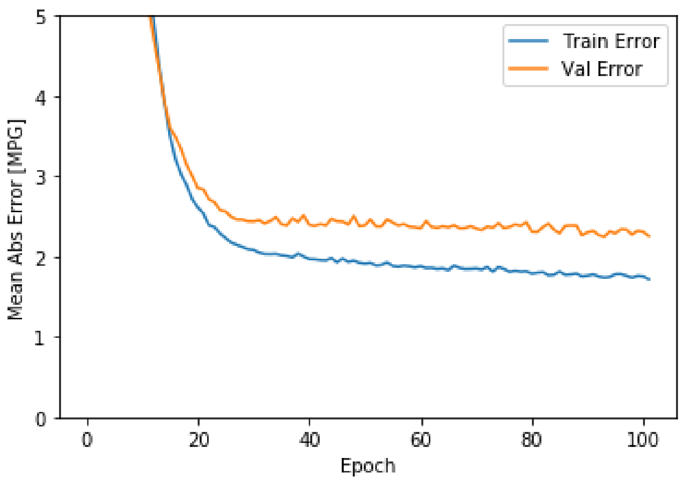
This graph shows little improvement, or even degradation in the validation error after about 100 epochs. Let's update the `model.fit` call to automatically stop training when the validation score doesn't improve. We'll use an *EarlyStopping callback* that tests a training condition for every epoch. If a set amount of epochs elapses without showing improvement, then automatically stop the training.

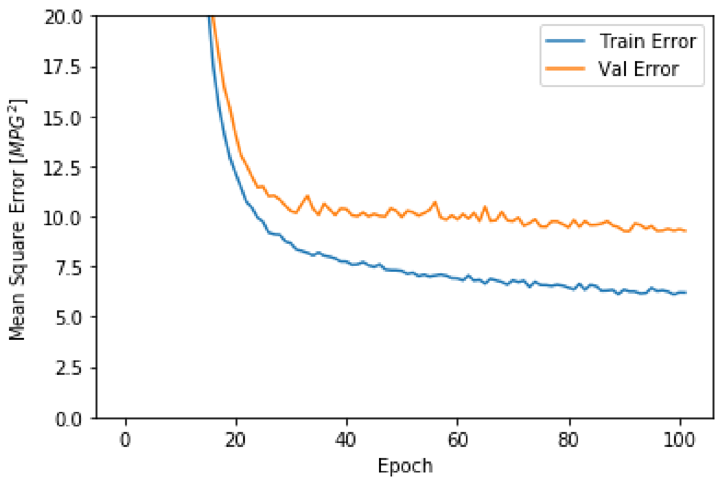
You can learn more about this callback here.

In [21]:

```
model = build_model()

# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(normed_train_data, train_labels, epochs=EPOCHS,
                    validation_split = 0.2, verbose=0, callbacks=[early_stop, PrintDot()
])

plot_history(history)
```

```
....................................................................................................
...........
..
```





The graph shows that on the validation set, the average error is usually around +/- 2 MPG. Is this good? We'll leave that decision up to you.

Let's see how well the model generalizes by using the test set, which we did not use when training the model. This tells us how well we can expect the model to predict when we use it in the real world.

In [22]:

```
loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=0)

print("Testing set Mean Abs Error: {:5.2f} MPG".format(mae))
```

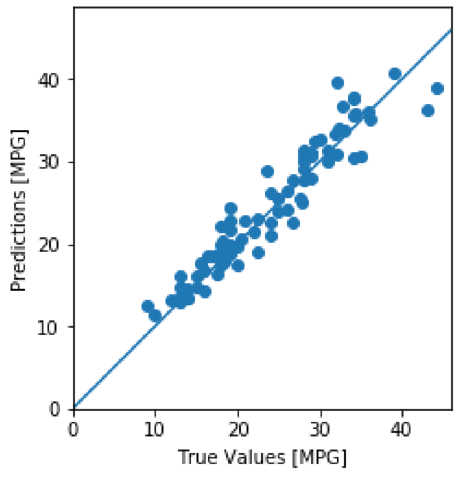
```
Testing set Mean Abs Error:  1.97 MPG
```

Make predictions

Finally, predict MPG values using data in the testing set:

In [23]:

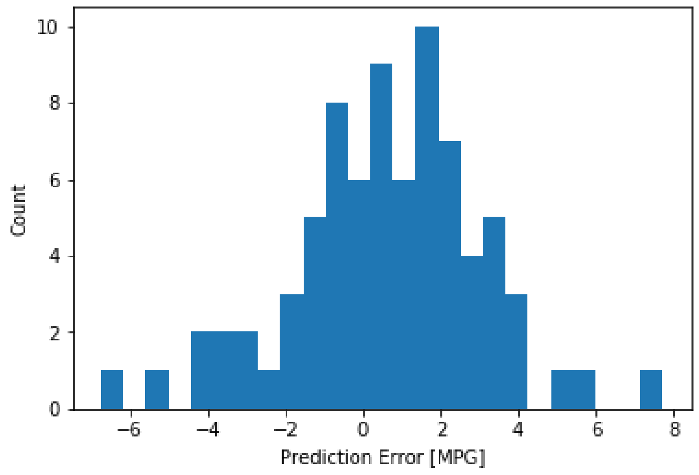
```
test_predictions = model.predict(normed_test_data).flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])
```



It looks like our model predicts reasonably well. Let's take a look at the error distribution.

In [24]:

```
error = test_predictions - test_labels
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [MPG]")
_ = plt.ylabel("Count")
```



It's not quite gaussian, but we might expect that because the number of samples is very small.

Conclusion

This notebook introduced a few techniques to handle a regression problem.

- Mean Squared Error (MSE) is a common loss function used for regression problems (different loss functions are used for classification problems).
- Similarly, evaluation metrics used for regression differ from classification. A common regression metric is Mean Absolute Error (MAE).
- When numeric input data features have values with different ranges, each feature should be scaled independently to the same range.
- If there is not much training data, one technique is to prefer a small network with few hidden layers to avoid overfitting.
- Early stopping is a useful technique to prevent overfitting.

- Early stopping is a useful technique to prevent overfitting.

License

Copyright 2018 The TensorFlow Authors.

In [25]:

In [26]:

Awesome Tutorials for Data Science with Python (TBU)

```
Common Data analysis and Machine learning tasks using Python
```

The Python Language

- Python 3 in one picture
- **Awesome Python**
- **Jargon from the functional programming world in simple terms!**
- **Dive Into Python**
- Learn Python Wiki on Reddit
- Learn 90% of Python in 90 Minutes
- Highest Voted Python Questions
- Python Basic Concepts
- Quick Reference to Python
- The Elements of Python Style
- **What does the yield keyword do in Python?**
- Parsing values from a JSON file in Python
- **Python Quora FAQs**

- time-complexity of various operations - list/dict - in current CPython
- Scripting in Python
  - Python Scripting Tutorial
  - Scripting with Python
  - **Can I use Python as a bash replacement?**

Useful Online Courses

- Learn Python (Codecademy)
- Free Interactive Course: Intro to Python for Data Science (DataCamp)
- Introduction to Computer Science and Programming Using Python (MIT)
- Python for Everybody
- Python Programming Essentials

Data Science with Python

- **Data Science IPython Notebooks**
- Awesome Python - Data Analysis
- Statistics
  - Statistics and Data Science
- **An Introduction to Scientific Python (and a Bit of the Maths Behind It) – NumPy**
- Data Analysis and IPython Notebooks
- Python for Data Science: Basic Concepts
- Pycon India 2015 Notes
- **5 important Python Data Science advancements of 2015**
- Data Exploration with Numpy cheat sheet
- Querying Craiglist with Python
- **An introduction to Numpy and Scipy**
- Create NBA Shot Charts
- PythoR- Python meets R
- **How do I learn data analysis with Python?**
- What are some interesting things to do with Python?
- **Which is better for data analysis: R or Python?**
- **Web scraping in Python**
- The Guide to Learning Python for Data Science
- Python For Data Science - A Cheat Sheet For Beginners
- Top voted Python data science questions
- Awesome Python - Data Visualization
- Awesome Python - Map Reduce

Pandas Library in Python

- Intro to pandas data structures
- Useful Pandas Cheatsheet
- An Introduction to Scientific Python – Pandas
- 10 minutes to Pandas
- Useful Pandas Snippets
- Timeseries analysis using Pandas
- Pandas Exercises - Practice your Pandas skills
- Grouping in Pandas
- **“Large data” work flows using pandas**
- Easier data analysis with pandas (video series)
- Pandas Basics Cheat Sheet
- Quick Operations on a Pandas DataFrame
  - Renaming Columns in Pandas (video)
  - Deleting Columns from pandas DataFrame (video)
  - Adding new Column to existing DataFrame
  - Add one Row in a pandas.DataFrame
  - Changing the order of DataFrame Columns
  - Changing data type of Columns (video)

    - Changing data type of Columns (video)
    - Getting a list of the column headers from a DataFrame
    - Converting list of dictionaries to Dataframe
    - Getting row count of pandas DataFrame
    - Most efficient way to loop through DataFrames
    - Deleting DataFrame row based on column value
    - Dropping a list of rows from Pandas DataFrame
    - Sorting a DataFrame or a single column
    - Filtering DataFrame rows by column value
    - Filtering DataFrame rows using multiple criteria
    - Dropping all non-numeric columns from a DataFrame
    - Counting and removing missing values
    - Selecting multiple rows and columns from a DataFrame
    - Reducing the size of a DataFrame

Machine Learning with Python

- AI, ML Related List
- Data Normalization in Python
- **Python Machine Learning Book**
- Table of Contents and Code Notebooks
- Machine Learning with scikit learn
- Machine Learning Algorithms Cheatsheet
- **How to compute precision, recall, accuracy and f1-score for the multiclass case with scikit learn?**
- One Hot Encoding for Machine learning in Python
- **Building a (semi) Autonomous Drone with Python**
- Awesome Python - Machine Learning
- Computer Vision
    - Awesome Python - Computer Vision

Scikit Learn

- scikit learn on Wikipedia
- **Introduction to machine learning with scikit-learn**, **Videos!**
- **A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library**
- **PyData Seattle 2015 Scikit-learn Tutorial**, sklearn_scipy2013
- SKLEARN BENCHMARKS: A centralized repository to report scikit-learn model performance across a variety of parameter settings and data sets, Report results of sklearn benchmarks at openml.org
- How to get most informative features for scikit-learn classifiers?
- **Code example to predict prices of Airbnb vacation rentals, using scikit-learn on Spark**
- **Machine Learning with scikit learn tutorial**
- Parallel and Large Scale Machine Learning with scikit-learn, Meetup
- Saving classifier to disk in scikit-learn

Linear Regression in Python

- Linear Regression in Python, Blog Post
- Linear Regression using Scikit Learn
- A friendly introduction to linear regression (using Python)
- Linear Regression Example in Python
- Regression analysis using Python StatsModels package
- Run an OLS regression with Pandas Data Frame

Logistic Regression in Python

- Logistic Regression with scikit learn
- Logistic Regression in Python
- Implementing the softmax function in Python
- **What is the inverse of regularization strength in Logistic Regression? How should it affect my code?**
- The Yhat Blog: Logistic Regression in Python

- Example of logistic regression in Python using scikit-learn
- TUTORIAL ON LOGISTIC REGRESSION AND OPTIMIZATION IN PYTHON
- Using Logistic Regression in Python for Data Science

k Nearest Neighbours in Python

- A good tutorial on implementing K Nearest Neighbors using scikit learn
- **Is it possible to specify your own distance function using scikit-learn K-Means Clustering?**
- Tutorial To Implement k-Nearest Neighbors in Python From Scratch
- Implementing your own k-nearest neighbour algorithm using Python
- knn Python implementation on StackOverflow
- kNN with big sparse matrices in Python
- Sklearn kNN usage with a user defined metric

Neural Networks in Python

- Implementing a Neural Network from scratch in Python , Code
- A Neural Network in 11 lines of Python
- Speeding up your Neural Network with Theano and the gpu , Code
- What is the best neural network library for Python?
- Recurrent Neural Net Tutorial in Python Part 1 , Part 2, Code
- PyBrain: modular Machine Learning Library for Python
- Neural Networks Tutorial – a Pathway to Deep Learning

Decision Trees in Python

- How to extract the decision rules from scikit-learn decision-tree?
- **How do I find which attributes my tree splits on, when using scikit-learn?**
- Quora: What is a good Python library for decision trees? , StackOverflow
- Building Decision Trees in Python
- Pure Python Decision Trees
- Building a decision tree from scratch in Python - a beginner's tutorial
- Using Python to Build and Use a Simple Decision Tree Classifier
- Decision trees in python with scikit-learn and pandas
- Code for simple decision tree in Python
- Lesson notebook: Regression and Classification Trees
- Discover structure behind data with decision trees

Random Forest with Python

- Getting Started with Random Forests: Titanic Competition on Kaggle , Python sample code
- RandomForestClassifier vs ExtraTreesClassifier in scikit learn
- Powerful Guide to learn Random Forest
- How are Feature Importances in RandomForestClassifier determined?
- Random forest interpretation with scikit-learn
- Random Forests in Python Tutorial
- Unbalanced classification using RandomForestClassifier in sklearn
- Random Forest with categorical features in sklearn
- How to output RandomForest Classifier from python?
- Lesson notebook: Ensembling, Bagging, and Random Forests

Support Vector Machine in Python

- Fastest SVM implementation usable in Python
- An example using python bindings for SVM library, LIBSVM
- What is the best SVM library usable from Python?
- How does sklearn.svm.svc's function predict_proba() work internally?
- Support vector machine in Python using libsvm example of features
- Linear SVC Machine learning SVM example with Python

- Understanding Support Vector Machine algorithm from examples (along with code)

NLP / Text Mining in Python

- **NLP with Python ORiley Book**, Python 3
- Awesome Python - NLP
- Awesome Python - Text Processing
- Text Analytics : Intro and Tokenization
- NLTK BOOK
- Elegant N-gram Generation in Python
- **Computing N Grams using Python**
- N-grams: Explanation + 2 applications
- NLP Tutorial with Python

Sentiment Analysis with Python

- A Comprehensive Guide to Sentiment Analysis
- Twitter-Sentiment-Analysis
- Basic Sentiment Analysis with Python
- What is the best way to do Sentiment Analysis with Python?
- How to Calculate Twitter Sentiment Using AlchemyAPI with Python
- Second Try: Sentiment Analysis in Python
- Sentiment Analysis with Python NLTK Text Classification
- Codes and Explanation
  - **Sentiment Analysis with bag-of-words**
  - **Sentiment Analysis with Naive Bayes**

Pickle: convert a python object into a character stream

- Python serialization - Why pickle?
- **Serializing Python Objects**, **Binary Files**
- What is Pickle in python ?
- How to cPickle dump and load separate dictionaries to the same file?
- **Understanding Pickling in Python**

AutoML

- TPOT: A Python tool for automating data science, GitHub repo

Regex Related

- RegExr
- Regex101
- Pythex
- How to use Regular Expressions (Regex) in Microsoft Excel both in-cell and loops
- Advanced Filters: Excel’s Amazing Alternative To Regex

Shell Scripting

- **Calling an external command in Python**
- **Running shell command from Python and capturing the output**
- **Can I use Python as a bash replacement?**
- Python Scripts as a Replacement for Bash Utility Scripts
- How to Write a Shell Script using Bash Shell in Ubuntu
- Red Hat Magazine | Python for Bash scripters: A well-kept secret
- Embed bash in python
- Bash2py: A Bash to Python Translator
- Beginners/BashScripting
- The Beginner’s Guide to Shell Scripting: The Basics

- Linux Shell Scripting Tutorial v1.05r3 A Beginner's handbook

Other good lists

- pycrumbs - Bits and bytes of Python from the Internet
- python github projects - Collect and classify python projects on Github
- python reference - Useful functions, tutorials, and other Python-related things
- pythonidae - Curated decibans of scientific programming resources in Python

Credits (Reference)

> - Regression: predict fuel efficiency
> - Ujjwal Karn
> - GitHub Awesome Lists Topic

License

Please *UPVOTE* my kernel if you like it or wanna fork it.

Feedback: If you have any ideas or you want any other content to be added to this curated list, please feel free to make any comments to make it better.

I am open to have your *feedback* for improving this *kernel*

Hope you enjoyed this kernel!

Thanks for visiting my *Kernel* and please *UPVOTE* to stay connected and follow up the *further updates!*