Introduction to Data Science

Python commands used

The following commands are used in the activities. The commands can be copied from this document (ctrl-C) and pasted into your code (ctrl-V).

Lesson 1: Introduction to Data Science

Import libraries

```
# import pandas for data analysis
import pandas as pd
# import seaborn for visualisations
import seaborn as sns
```

Import a csv file

```
# import the csv file to a data set called weather_data
weather_data = pd.read_csv('../input/weather-data-edexcel-large-data-set/all-stations-uk.csv')
```

Display information about the data set

```
# display the first 6 rows of the data set
weather_data.head(6)
```

`.info()` is particularly useful as the exact field names can be copied into other commands.

```
# explore the data types
weather_data.info()
```

Display the summary statistics for a feature

```
# calculate the summary statistics
weather_data['Daily Mean Temperature'].describe()
```

Display a boxplot

```
# display a boxplot for a feature using catplot from Seaborn
sns.catplot(data=weather_data, kind='box', x='Daily Mean Temperature');
```

Displaythe grouped statistics for a feature

```
# calculate the grouped summary statistics
weather_data.groupby('Station')['Daily Mean Temperature'].describe()
```

Display grouped boxplots

```
# display a boxplot for a feature using catplot from Seaborn. Note the y variable groups by the categorical feature
sns.catplot(data=weather_data, kind='box', x='Daily Mean Temperature', y='Station', aspect=2);
```

Lesson 2: Pre-processing Data

Cleaning data: remove commas

```
# commas in the ‘In employment’ feature are removed by replacing them with an empty string
travel_2011_data['In employment value'] = travel_2011_data['In employment'].str.replace(',', '').astype('int')
```

Create a derived field (e.g. percentage)

```
# The percentage is calculated and stored in a new feature: Bicycle percent
travel_2011_data['Bicycle percent'] = travel_2011_data['Bicycle value'] / travel_2011_data['In employment value'] * 100
```

Filtering: Create a copy of a data set with rows removed

```
# a new copy of the data set is created containing only cars with cars_data['Mass']>0 and cars_data['CO2']>0
cars_data2 = cars_data[(cars_data['Mass'] > 0) & (cars_data['CO2'] > 0)].copy()
```

Filtering: create a new dataset based on a subset of an existing one

```
# a new data set is created by copying only the rows where the Propulsion Type is Petrol
petrol_data = cars_data2[cars_data2['PropulsionType'] == 'Petrol'].copy()
```

Create a text feature for a category stored as numbers

```
# .replace() uses a colon to indicate what is replaced and commas to separate the items
cars_data2['PropulsionType'] = cars_data2['PropulsionTypeId'].replace({1: 'Petrol', 2: 'Diesel', 3: 'Electric'})
```

Lesson 3: Data presentation and visualisation

Display a box plot

```
# creating a box plot - define the data and the x-variable
sns.catplot(data=weather_data, kind='box', x='Daily Mean Temperature');
```

Display a box plot grouped by a categorical feature

```
# create a box plot with a category on the y-axis. Note that aspect=2 gives a plot twice as wide as it is high
sns.catplot(data=weather_data, kind='box', x='Daily Mean Temperature', y='Station', aspect=2);
```

Display a box plot grouped by a categorical feature and split by another categorical feature (using *hue*)

```
# create a box plot with a category on the y-axis, colour-coded by another category
sns.catplot(data=weather_data, kind='box', x='Daily Mean Temperature', y='Station', hue='Year', aspect=2);
```

Display a density plot grouped by a categorical feature

```
# create a strip plot with a category on the y-axis
sns.catplot(data=weather_data, kind='strip', x='Daily Mean Temperature', y='Station', aspect=2);
```

Display a violin plot grouped by a categorical feature and split by a binary feature for hue

```
# generate a split violin plot with a category for y and a binary for hue
sns.catplot(data=weather_data, kind='violin', x='Daily Mean Temperature', y='Station', hue='Year', split=True,  aspect=2);
```

Display a histogram

```
# plot a histogram
sns.displot(data=weather_data, x='Daily Mean Temperature', aspect=2);
```

Display a histogram grouped by a categorical feature in different columns

```
# plot histograms split by a category over columns
sns.displot(data=weather_data, x='Daily Mean Temperature', col='Station');
```

Display a kernel density estimation (KDE)

```
# kind='kde' estimates the shape of a continuous distribution, categorised using hue
sns.displot(data=weather_data, kind='kde', x='Daily Mean Temperature', hue='Station', aspect=2 );
```

Display a scatter diagram

```
# create a scatter plot of life expectancy against GDP
sns.relplot(data=countries_data, x='GDP per capita (US$)', y='Life expectancy at birth 2010', aspect=2);
```

Display a scatter diagram with a third (categorical) variable shown by the colour

```
# create a scatter plot of life expectancy against GDP, colour-coded by region
sns.relplot(data=countries_data, x='GDP per capita (US$)', y='Life expectancy at birth 2010', hue='Region', aspect=2);
```

Display a scatter diagram with a third (numerical) variable shown by the size

```
# create a scatter plot of life expectancy against GDP, with size as physician density, sizes=(30,150) sets size scale
sns.relplot(data=countries_data, x='GDP', y='Life expectancy', size='physician density', sizes=(30, 150), aspect=2);
```

Display a hexagonal bin plot

```
# create a hexagonal bin plot
sns.jointplot(data=countries_data, kind='hex', x='GDP per capita (US$)', y='Life expectancy at birth 2010');
```

Display a time series plot

```
# time series of median house prices by national region. Note lw=3 gives thicker lines
sns.relplot(data=national_data, kind='line', x='Year', y='MEDIAN HOUSE PRICE (£)', hue='Area', lw=3, aspect=2);
```

Display a time series plot with the y-axis starting at 0

```
# define fig as a time series of median house prices with a y-axis that starts at zero
fig = sns.relplot(data=national_data, kind='line', x='Year', y='MEDIAN HOUSE PRICE (£)', hue='Area', lw=3, aspect=2);
# set the y-axis of fig to start at zero, and let seaborn decide the upper limit
fig.set(ylim=(0, None));
```

Lesson 5: Introduction to machine learning

Build and measure a linear regression model

```
# define the input feature(s) and output (or target) feature
input_features = ['Mass']
input_data = petrol_data[input_features]
target_data = petrol_data['CO2']

# use the train_test_split command to create training and testing data
input_train, input_test, target_train, target_test = train_test_split(input_data, target_data, train_size=0.75, random_state=1)

# create the model
linear_model = LinearRegression().fit(input_train, target_train)

# display the model
print('Coefficients: ', (linear_model.coef_).round(3))
print('Intercept: ', (linear_model.intercept_).round(3))

# create a list of the predictions and calculate R² by comparing with the testing data
target_pred = linear_model.predict(input_test)
print('R²: ',r2_score(target_test, target_pred).round(3))
```

Create a plot_decision_tree command

```
# create a plot_decision_tree command for displaying the decision tree
def plot_decision_tree(model,size=14):
    fig, ax = plt.subplots(figsize=(18,8))
    plot_tree(model, filled=True, impurity=False, label='root', feature_names=input_features, proportion=True,
              class_names=["No", "Yes"], ax=ax, fontsize=size)
    plt.show()
```

Build and measure a binary classification model

```
# define the input feature(s) and output (or target) feature
input_features = ['bill_length_mm']
input_data = penguin_data[input_features]
target_data = penguin_data['gentoo']

# use the train_test_split command to create training and testing data
input_train, input_test, target_train, target_test = train_test_split(input_data, target_data, train_size=0.75, random_state=1)

# create the model
tree_model = DecisionTreeClassifier(max_depth=1).fit(input_train, target_train)

# display the model
plot_decision_tree(tree_model)

# create a list of the predictions and calculate the accuracy by comparing with the testing data
target_pred = tree_model.predict(input_test)
print("Accuracy: ",(100*accuracy_score(target_test, target_pred)).round(1),"%")
```