Import panda as pd

X=Pd.read_csv('file path')

This generates a "Panda DataFrame type" object X which can be accessed using panda based commands. X has all the contents of the csv file.

1) X.describe()

This tells you stuff about the Table and its columns. Notice how it's a *method*.

It gives No of entries, Mean, Standard Dviation, Min, Max ,25,50,75 percentiles. The 4 quartiles, basically.

2) X.dropna(axis=0)

Drop na can be thought of as drop if something is not available. If we have rows of data that are incomplete, it will complicate the training of an algorithm using that data. In order to circumvent this, we just drop any rows of data that contain null values. Axis =0 refers to Rows. Axis =1 refers to columns. So, if we used acis=1, all columns that have empty values will be deleted. In my example, it's not the intended effect.

The dropna method can use various parameters to selectively destroy something whenever na values are encountered. I will not presently attempt to meddle with such attributes.

3) X.head()

This method gives the first 5 rows of your table.

1) C=X.columns

This returns a "panda Index type" object which I'm storing in C. This is an *attribute*, NOT a method.

The object contains the names of all the columns.

2) Y=X.columnname

We can obtain a single column of data by calling this attribute. The object Y so generated is a "panda Series type" object.

3) Y=X[["cname1","cname2","cname3"…]]

Notice the double brackets. This is typically done as X[ColumnsList] where ColumnsList is a list object containing the names of the columns you wish to look at. Notice how the syntax has no dots, and is pretty similar to list slicing.

This returns another DataFrame Y, a subset of X, if you will.

A simple Decision Tree Implementation:

```
# X=subset of data apart from the value to be predicted. This is like Area of
House, No of bedrooms etc.
# y=column of values to be predicted. This is like price of house.
# Testcases= data without y given beforehand. We need to run the model on this
data to predict the price of house or whatever.

from sklearn.tree import DecisionTreeRegressor
X_model= DecisionTreeRegressor(random_state=1)
X_model.fit(X, y)
X_model.predict(Testcases)
```

These commands are rather English like, and there is no complexity for us to handle. The random_state parameter is entirely optional. Typically the regressor starts off randomly. We are giving a starting point here just to get the same results every time the code gets executed.

Checking if our model is worth anything can be done by predicting the outcomes of another dataset. If we don't have bonus data lying around we can use the train_test_split function to split the data into a training dataset and a validation dataset, which we can then use to validate the performance of our model.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)

X_model = DecisionTreeRegressor()
X_model.fit(train_X, train_y)

val_predictions = melbourne_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))
```

MAE is fine, but I'm used to MSE after eons of statistics.

DesicionTreeRegressor(max_leaf_nodes=150) This parameter, max lead nodes, helps us prevent overfitting. Checking various values of this and picking the optimal one is a decent strategy.

This is the first time I'm hearing about decision trees, and having seen regressors that seemed more theoretically flawless, and mathematically grounded, I havent taken much of a liking to it.

A more sophisticated version of the decision tree is a Random forest.

```
from sklearn.ensemble import RandomForestRegressor
forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
```

Using it via sklearn is no big deal, they have already done all the work for us.

Things to do to make your model more awesome:

- Check dependence on prediction value
- Do Kfold checking. Splitting the training data 5 ways helps a balanced result.
- Check for inconsistency in the input data eg dd/mm/yyyy AND mm/dd/yyyy may be in the same column, different units may be used in the same column etc.
- Average out multiple predictions from different models.
- Try predicting something other than the direct objective.
- Do Feature importance checking. Maybe only certain parameters need to be given high priority.
- Club Test and train when you want to make modifications like normalisation, remapping of categorical variables etc so that you don't have to do any double work.