

TO START

```
# IMPORT DATA LIBRARIES
import pandas as pd
import numpy as np

# IMPORT VIS LIBRARIES
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

# IMPORT MODELLING LIBRARIES
from sklearn.model_selection import
train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

PRELIMINARY OPERATIONS

<code>df = pd.read_csv('data.csv')</code>	read data
<code>df.head()</code>	check head df
<code>df.info()</code>	check info df
<code>df.describe()</code>	check stats df
<code>df.columns</code>	check col names

VISUALISE DATA

<code>sns.pairplot(df)</code>	pairplot
<code>sns.distplot(df['Y'])</code>	distribution plot
<code>sns.heatmap(df.corr(), annot=True)</code>	heatmap with values

TRAIN MODEL

❏ CREATE X and y -----

<code>X = df[['col1', 'col2', etc.]]</code>	create df features
<code>y = df['col']</code>	create df var to predict

❏ SPLIT DATASET -----

<code>X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)</code>	split df in train and test df
---	-------------------------------

📊 FIT THE MODEL -----

<code>lm = LinearRegression()</code>	instantiate model
<code>lm.fit(X_train, y_train)</code>	train/fit the model

👁 SHOW RESULTS -----

TRAIN MODEL (cont)

<code>lm.intercept_</code>	show intercept
<code>lm.coef_</code>	show coefficients
<code>coeff_df = pd.DataFrame(lm.coef_, X.columns, columns=['Coeff'])</code>	create coeff df

pd.DataFrame: `pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)`. **data** = values, **index** = name index, **columns** = name column. This could be useful just to interpret the coefficient of the regression.

MAKE PREDICTIONS

<code>predictions = lm.predict(X_test)</code>	create predictions
<code>plt.scatter(y_test, predictions)*</code>	plot predictions
<code>sns.distplot((y_test-predictions), bins=50)*</code>	distplot of residuals

scatter: this graph shows the difference between actual values and the values predicted by the model we trained. It should resemble as much as possible a **diagonal line**.

distplot: this graph shows the distributions of the residual errors, that is, the difference between the actual values minus the predicted values; it should result in an as much as possible **normal distribution**. If not, maybe change model!

EVALUATION METRICS

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

MAE is the easiest to understand, because it's the average error.

MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.

RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units.

