Model Development

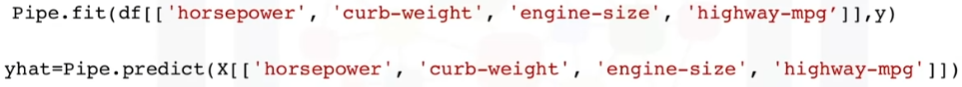
* Simple and Multiple Linear Regression
  + Simple – 1 independent variable to make a prediction (SLR)
    - Predictor X for Target Y
    - Y= b0 + b1x (b0 the intercept and b1 the slope)
    - Code
      * **From sklearn.linear\_model import LinearRegression**
      * **lm = LinearRegression()**
      * **x = df[[‘name’]]**
      * **y = df[[‘name’]]**
      * **lm.fit(x, y)**
      * **yhat=lm.predict (x)**
      * **yhat[0:endrange]**
    - Find intercept
      * **.intercept\_**
    - Find Slope
      * **.coef\_**
  + Text

    Description automatically generatedMultiple – more than 1 independent variable to make a prediction (MLR)
    - With multiple store all predictor variables in as a single variable
      * **Z = df[[‘name1’, ‘name2’, ‘name3’]]**
      * **Lm.fit(Z, df[[‘dependent’]])**
      * **Yhat=lm.predict(x)**
    - Plotting multiple linear regression example
      * **plt.figure(figsize=(width, height))**
      * **ax1 = sns.distplot(df['price'], hist=False, color="r", label="Actual Value")**
      * **sns.distplot(Y\_hat, hist=False, color="b", label="Fitted Values" , ax=ax1)**
      * **plt.title('Actual vs Fitted Values for Price')**
      * **plt.xlabel('Price (in dollars)')**
      * **plt.ylabel('Proportion of Cars')**
      * **plt.show()**
      * **plt.close()**
* Model Evaluation using Visualization
  + Use regression plots from seaborn to evaluate
    - Code:
      * **Import seaborn as sns**
      * **sns.regplot(x=”highway-mpg”, y= “price”, data=df)**
      * **Plt.ylim(0,)**
  + **Scatter chart

    Description automatically generated with medium confidence**Residual Plot
    - Plotting the predicted value against the actual value.
    - The average of the entire plot should have a 0 mean.
    - Chart, scatter chart

      Description automatically generatedPlot on the right suggest that linear plots are appropriate.
    - Example on where linear model are not appropriate.
    - Ex:
      * **width = 12**
      * **height = 10**
      * **plt.figure(figsize=(width, height))**
      * **sns.residplot(df['highway-mpg'], df['price'])**
      * **plt.show()**
  + Creating Residual Plots
    - Code
      * **Import seaborn as sns**
      * **Sns.residplot(df[‘independent var’, df[‘dependent var’]]**
  + Distribution Plots
    - Code
      * **Import seaborn as sns**
      * **Ax1 = sns.distplot(df[‘price’], hist= False, color=”r”, label= “Actual Value”)**
      * **Sns.distplot(yhat, hist=False, color=”b”, label=”Fitted Values”, ax=ax1)**
    - Code
      * **Import seaborn as sns**
      * **yhat\_test1=poly1.predict(x\_test\_pr1)**
      * **Title='Distribution Plot of Predicted Value Using Test Data vs Data Distribution of Test Data'**
      * **DistributionPlot(y\_test, yhat\_test1, "Actual Values (Test)", "Predicted Values (Test)", Title)**
* Polynomial Regression and Pipelines
  + Polynomial Regression
    - Quadratic – 2nd Order
    - Cubic – 3rd Order
    - Code
      * **x = df['highway-mpg']**
      * **y = df['price']**
      * **f=np.polyfit(x,y,3) \*3 is the degree of the polynomial**
      * **p=np.poly1d(f)**
      * **p**
      * **\***to plot: **PlotPolly(p, x, y, 'highway-mpg')**
  + Preprocessing for more than one dimension
    - Code
      * **From sklearn.preprocessing import PolynomialFeatures**
      * **pr = PolynomialFeatures(degree = 2, include\_bias =False)**
      * **x\_polly=pr.fit\_transform(x[[‘horsepower’,’curb-weight’]]**
  + Normalizing each feature simultaneously
    - Code
      * **From sklearn.preprocessing import StandardScaler**
      * **SCALE=StandardScaler()**
      * **SCALE.fit(x\_data[[‘horsepower’,’highway-mpg’]])**
      * **X\_scale=SCALE.transform(x\_data[[‘horsepower’,’highway-mpg’]])**
  + Diagram

    Description automatically generatedPipelines
    - Text

      Description automatically generatedImport these modules
    - Create a list of tuples
      * **Input=[('scale',StandardScaler()),('polynomial', PolynomialFeatures(include\_bias=False)),('model',LinearRegression())]**
    - Create pipeline contructor
      * **pipe=Pipeline(Input)**
    - Train the pipeline Object
    - Example: Standardizing pipeline.
      * **Input=[('scale',StandardScaler()), ('polynomial', PolynomialFeatures(include\_bias=False)), ('model',LinearRegression())]**
      * **pipe=Pipeline(Input)**
      * **Z = Z.astype(float)**
      * **pipe.fit(Z,y)**
      * **ypipe=pipe.predict(Z)**
      * **ypipe[0:4]**
* R-squared and MSE for in-sample evaluation
  + Numerically evaluate models.
    - Mean Squared Error (MSE)
      * Formular: sum of all (Actual – Predicted)^2 / Total number of samples.
      * Code:
        + **from sklearn.metrics import mean\_squared\_error**
        + **mean\_squared\_error(df[‘price’], Y\_predict\_simple\_fit)**
    - Text

      Description automatically generated with medium confidenceR-Squared
      * To calculate R^2 (should be between 0-1)
        + Code:

**X = df[[‘var 1’]]**

**y = df[[‘var 2’]]**

**lm.fit(x, Y)**

**lm.score(x,y)**

* + - * + **Example**

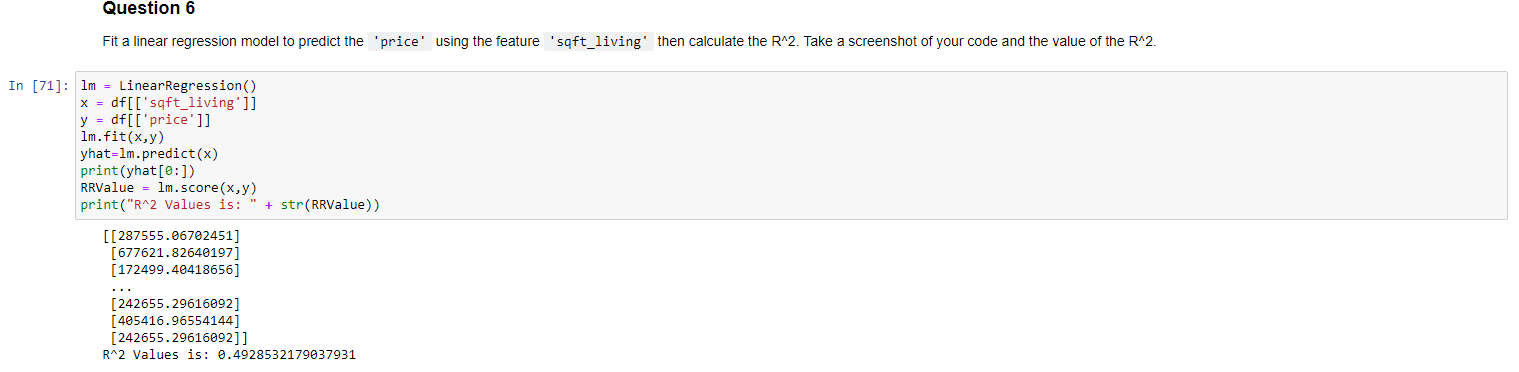
**from sklearn.metrics import r2\_score**

**r\_squared = r2\_score(y, p(x))**

**print('The R-square value is: ', r\_squared)**

**mean\_squared\_error(df['price'], p(x)**

* Prediction and decision Making
  + Getting a single prediction
    - **Lm.fit(df[‘highway-mpg’],df[‘prices’])**
    - **Lm.predict(np.array(30.0),reshape(-1,1))**
    - \*getting coefficient with: **lm.coef**
  + Getting a sequence of values
    - **Code**
      * **Import numpy as np**
      * **New\_input=np.arrange(start,end,stepSize).reshape(-1,1)**
      * **Yhat=lm.predict(new\_input)**

**Graphical user interface, text, application, email

Description automatically generated**