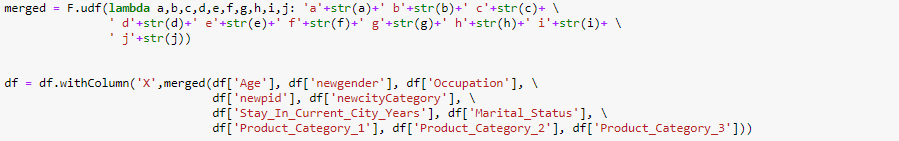
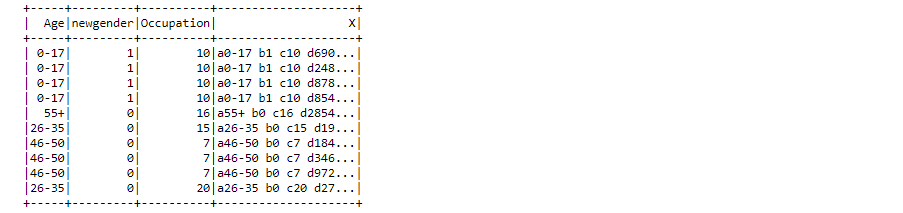
**A Logistic Regression classifier has been selected for this project. The classifier takes a vector as an input, and outputs the label of the bucket of the predicted purchase price. The input feature vector has been made by encoding the columns of the data frame into a sparse ‘Bag of Words’ vector. The output of the model is the label that maps to the predicted purchase price bucket.**

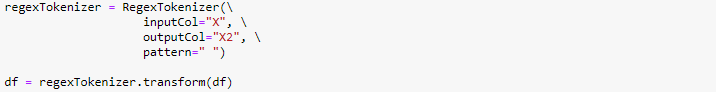
Creation of ‘sentence of words’:



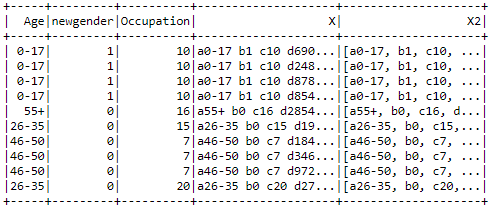
To create a bag of words, each of the possible values of each column are concatenated with a column id as a ‘word’ and made into strings a.k.a. a ‘sentence’ of words. For example: the word for value 10 on column ‘Occupation’ would be c10 (The column id of ‘Occupation’ column is ‘c’) and would be a part of a ‘sentence’ of all ‘words’ in the row. This process is the natural language processing equivalent of having a sentence of words. Here, column ‘X’ has the sentence generated by the previous columns.



Tokenizing ‘words’:

****

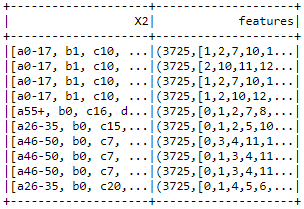
The sentence needs to be converted into a vector of words to so that it can be processed further. For this, Spark’s RegexTokenizer is used. It takes a regex pattern as a parameter and creates a new column with the vector. Since the words in the ‘sentence’ in column ‘X’ are separated by space, the regex given to split the words is simply “ ” which separates words in the sentence by splitting at every space. The output column is ‘X2’



Creating the sparse vector:

****

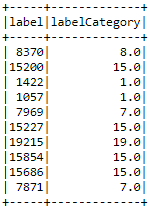
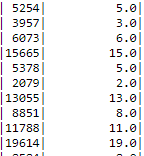
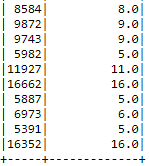
The sparse ‘Bag of Words’ vector is the input to the Logistic Regression Model. It is created by using spark’s implementation of ‘Bag of Words’ which is the CountVectorizer. A model is configured with the input column X2 and output column ‘features’. The sparse vector generated in this column (‘features’) can uniquely identify each possible configuration of the set of columns. Two rows will have the same vector in the ‘features’ column if and only if all their relevant columns’ values are the same.



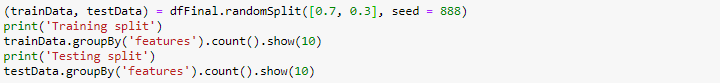
Converting purchase price into labeled buckets:

****

Logistic Regression attempts to predict a discrete label from a set of categorical labels. Due to this, the purchase values are converted into labels. This is done by creating buckets of purchase values. Increasing the bucket size means there will be lesser number of labels to predict from, and the prediction accuracy of the model increases. However, this also means that the predicted purchase range is wider, which decreases the usefulness of the prediction. To strike a balance between the two, the bucket size was chosen based on the bucket size that gave least ‘absolute prediction error’ (The average error in the predicted price bucket’s median w.r.t. the actual purchase price)

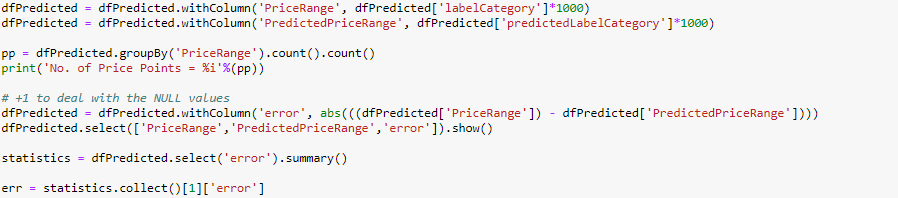
  

Training and testing the logistic regression model:

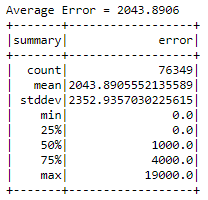
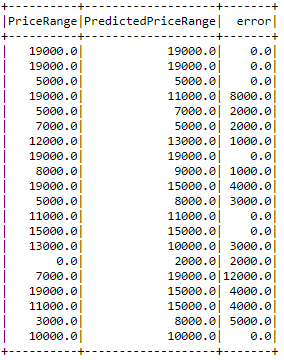




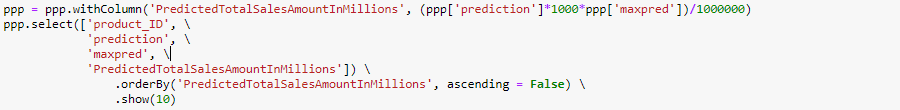
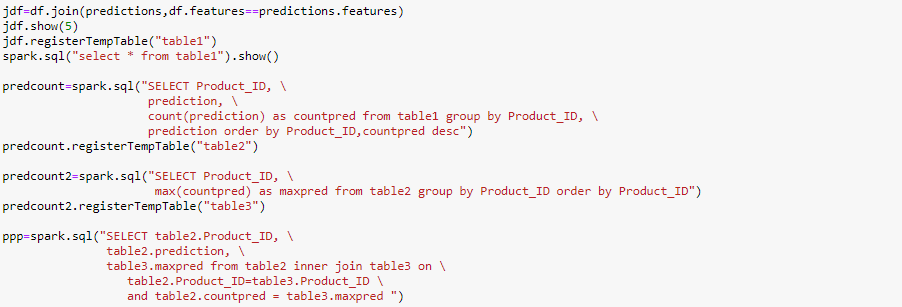




The machine learning model trains on the training split which is a subset of the entire dataset. While testing its accuracy, it cannot be tested on the same data. For this, a test split is created with the data not used for training. A 70/30 training/testing split has been made by using spark data frames’ randomSplit() functionality. Once the model is trained, its error rate statistics are calculated and evaluated. In order to calculate the error rate of these categorical values, the ‘least absolute prediction error’ is found as described earlier and evaluated.



Prediction of total sales:



In order to predict the estimated sales of each product, we predict the purchase prices using the trained model. **[PUT LOGIC FOR CREATING MAX PRED]**. The predicted total sales is the purchase price of a product times its ‘maxpred’ value. For this calculation, the lower end of the purchase price range is used as the purchase price (to ensure a conservative estimate). The predicted sales row is calculated and divided by 106 to give the predicted total sales amount in millions.

