Predict that the customer repeat-purchased the product

### Let’s begin with exploration:

Importing libraries and the data set:

* numpy
* matplotlib
* pandas
* sklearn

After importing the library, you read the dataset using function read\_csv(). This is how the code looks like till this stage:

import pandas as pd

import numpy as np

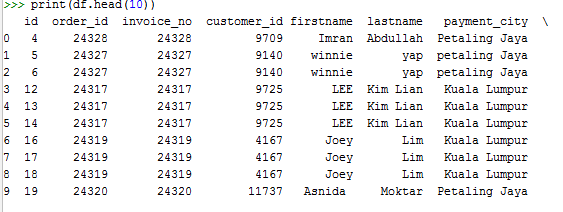
import matplotlib as plt

df = pd.read\_csv(**"D:\Machine learning Work\Data\Sample-Dataset1.csv"**, low\_memory=**False**) #Reading the dataset in a dataframe using Pandas

### Quick Data Exploration

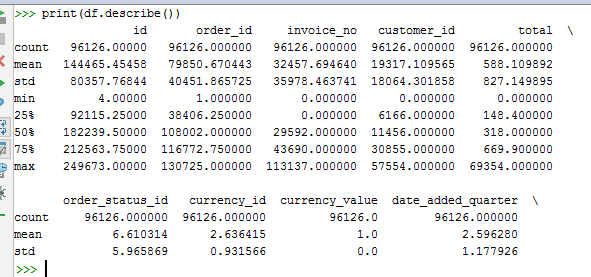
Once you have read the dataset, you can have a look at few top rows by using the function head()

df.head(10)



Next, you can look at summary of numerical fields by using describe() function

df.describe()

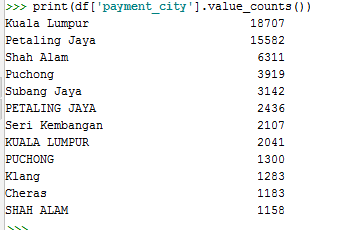


describe() function would provide count, mean, standard deviation (std), min, quartiles and max in its output

Please note that we can get an idea of a possible skew in the data by comparing the mean to the median, i.e. the 50% figure.

For the non-numerical values (e.g. Payment city, analysis\_category\_code etc.), we can look at frequency distribution to understand whether they make sense or not. The frequency table can be printed by following command:

df['**payment\_city**’].value\_counts()

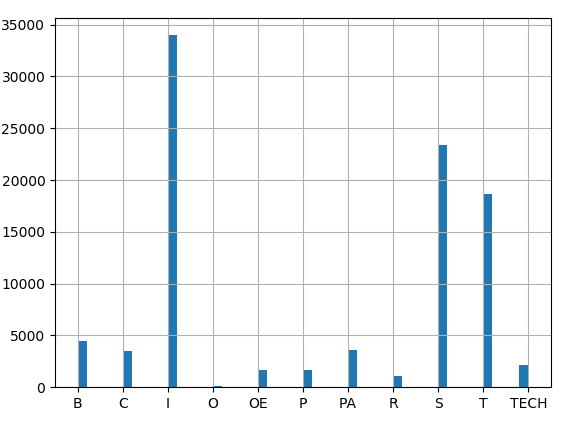


### Distribution analysis

Now that we are familiar with basic data characteristics, let us study distribution of various variables. Let us start with numeric variables – namely analysis\_category\_code

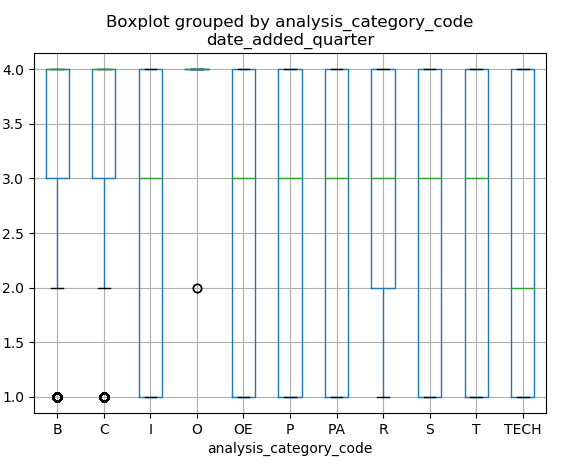
Lets start by plotting the histogram of analysis\_category\_code using the following commands:

df[**'analysis\_category\_code'**].hist(bins=50)



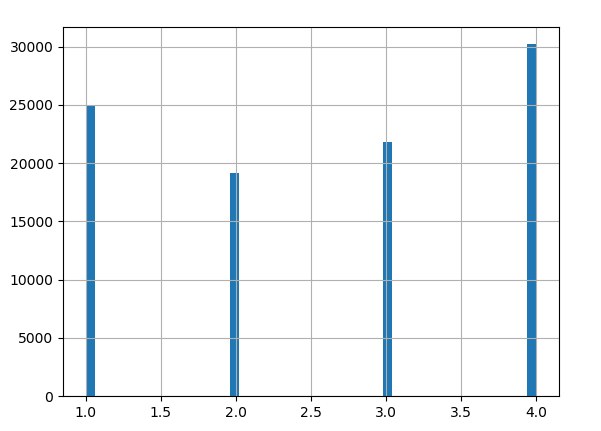
Next, we look at box plots to understand the distributions. Box plot for fare can be plotted by:

df.boxplot(column='date\_added\_quarter', by = 'analysis\_category\_code')



Now, Let’s look at the histogram and boxplot of date added quarter using the following command:

df[**'date\_added\_quarter'**].hist(bins=50)



### Data munging:

While our exploration of the data, we found a few problems in the data set, which needs to be solved before the data is ready for a good model. This exercise is typically referred as “Data Munging”. Here are the problems, we are already aware of:

1. There are missing values in some variables. We should estimate those values wisely depending on the amount of missing values and the expected importance of variables.
2. While looking at the distributions, we saw that Category code and date added quater seemed to contain extreme values at either end. Though they might make intuitive sense, but should be treated appropriately.

### Check missing values in the dataset

Let us look at missing values in all the variables because most of the models don’t work with missing data and even if they do, imputing them helps more often than not. So, let us check the number of nulls / NaNs in the dataset

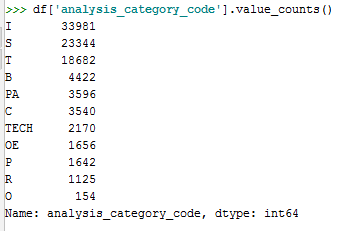
### df.apply(lambda x: sum(x.isnull()),axis=0)

### 

### 

Filling the missing values in the Category field and I think it’s important for me to predict by category.

df[**'analysis\_category\_code'**].value\_counts()



## Building a Predictive Model:

After, we have made the data useful for modeling, let’s now look at the python code to create a predictive model on our data set. Skicit-Learn (sklearn) is the most commonly used library in Python for this purpose and we will follow the trail.

Since, sklearn requires all inputs to be numeric, we should convert all our categorical variables into numeric by encoding the categories. This can be done using the following code:

After careful analysis and evaluation, below fields are to be included in the model preparation as rest other are simply not useful and important.

**'customer\_id','total','order\_status\_id','date\_added\_month','date\_added\_year','total\_qty',**

**'product\_quantity','product\_total','analysis\_category\_coded','Pred\_Q1','Pred\_Q2',**

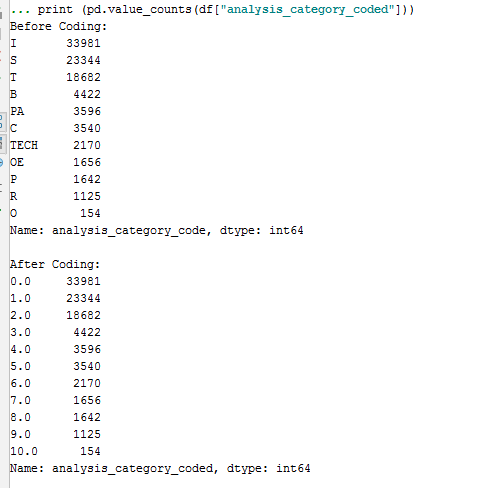
**'Pred\_Q3','Pred\_Q4'**

**Problem Statement: Predicting that the customer will buy the product in the quarter by category wise.**

Hence the fields **'Pred\_Q1','Pred\_Q2','Pred\_Q3','Pred\_Q4'** are derived from the date\_added\_quater, if Q=1, then Pred\_Q1 is 1 else 0 and other follow same respectively.

Before proceding to model, we need to convert the categorical values in the field **'analysis\_category\_coded' to the nominal or numerical**

*Coding nominal data  
# Define a generic function using Pandas replace function***def** coding(col, codeDict):  
 colCoded = pd.Series(col, copy=**True**)  
 **for** key, value **in** codeDict.items():  
 colCoded.replace(key, value, inplace=**True**)  
 **return** colCoded  
  
  
*#Coding analysis\_category\_code as I=0, S=1 etc:*print (**'Before Coding:'**)  
print (pd.value\_counts(df[**"analysis\_category\_code"**]))  
df[**"analysis\_category\_coded"**] = coding(df[**"analysis\_category\_code"**], {**'I'**:0,**'S'**:1,**'T'**:2,**'B'**:3,**'PA'**:4,**'C'**:5,**'TECH'**:6,**'OE'**:7,**'P'**:8,**'R'**:9,**'O'**:10 })  
print (**'\nAfter Coding:'**)  
print (pd.value\_counts(df[**"analysis\_category\_coded"**]))



Next, we will import the required modules. Then we will define a generic classification function, which takes a model as input and determines the Accuracy and Cross-Validation scores.

*# Generic function for making a classification model and accessing performance:***def** classification\_model(model, data, predictors, outcome):  
 *# Fit the model:* model.fit(data[predictors], data[outcome])  
  
 *# Make predictions on training set:* predictions = model.predict(data[predictors])  
  
 *# Print accuracy* accuracy = metrics.accuracy\_score(predictions, data[outcome])  
 print(**"Accuracy : %s"** % **"{0:.3%}"**.format(accuracy))  
  
 *# Perform k-fold cross-validation with 5 folds* kf = KFold(data.shape[0], n\_folds=5)  
 error = []  
 **for** train, test **in** kf:  
 *# Filter training data* train\_predictors = (data[predictors].iloc[train, :])  
  
 *# The target we're using to train the algorithm.* train\_target = data[outcome].iloc[train]  
  
 *# Training the algorithm using the predictors and target.* model.fit(train\_predictors, train\_target)  
  
 *# Record error from each cross-validation run* error.append(model.score(data[predictors].iloc[test, :], data[outcome].iloc[test]))  
  
 print(**"Cross-Validation Score : %s"** % **"{0:.3%}"**.format(np.mean(error)))  
  
 *# Fit the model again so that it can be refered outside the function:* model.fit(data[predictors], data[outcome])

### Logistic Regression

Let’s make our first Logistic Regression model. One way would be to take all the variables into the model but this might result in overfitting (don’t worry if you’re unaware of this terminology yet). In simple words, taking all variables might result in the model understanding complex relations specific to the data and will not generalize well.

So let’s make our first model with ‘analysis\_category\_coded’.

*#Logistic Regression.*outcome\_var = **'Pred\_Q1'**model = LogisticRegression()  
predictor\_var = [**'analysis\_category\_coded'**]  
classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 74.078% Cross-Validation Score : 74.078%

*#We can try different combination of variables:*predictor\_var = [**'analysis\_category\_coded'**,**'total'**,**'date\_added\_month'**,**'date\_added\_year'**]  
classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 96.234% Cross-Validation Score : 94.434%

Generally we expect the accuracy to increase on adding variables. But this is a more challenging case. The accuracy and cross-validation score are not getting impacted by less important variables. Category Code is dominating the mode. We have two options now:

1. Feature Engineering: dereive new information and try to predict those. I will leave this to your creativity.
2. Better modeling techniques. Let’s explore this next.

### Decision Tree

Decision tree is another method for making a predictive model. It is known to provide higher accuracy than logistic regression model.

*#Decision Tree*model = DecisionTreeClassifier()  
predictor\_var = [**'analysis\_category\_coded'**]  
classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 74.078% Cross-Validation Score : 74.078%

Here the model based on categorical variables is unable to have an impact because Category Code is dominating over them. Let’s try a few numerical variables:

*#We can try different combination of variables:*predictor\_var = [**'analysis\_category\_coded'**,**'total'**,**'date\_added\_month'**,**'date\_added\_year'**]  
classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 100.000% Cross-Validation Score : 100.000%

Here we observed that although the accuracy went up on adding variables, the cross-validation error went down. This is the result of model over-fitting the data. Let’s try an even more sophisticated algorithm and see if it helps:

### Random Forest

Random forest is another algorithm for solving the classification problem.

An advantage with Random Forest is that we can make it work with all the features and it returns a feature importance matrix which can be used to select features.

*#Random Forest*model = RandomForestClassifier(n\_estimators=100)  
predictor\_var = [**'analysis\_category\_coded'**]  
classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 74.078% Cross-Validation Score : 74.078%

*#We can try different combination of variables:*predictor\_var = [**'customer\_id'**,**'total'**,**'order\_status\_id'**,**'date\_added\_month'**,**'date\_added\_year'**,**'total\_qty'**,  
 **'product\_quantity'**,**'product\_total'**,**'Pred\_Q2'**,**'Pred\_Q3'**,**'Pred\_Q4'**,**'analysis\_category\_coded'**]  
classification\_model(model, df,predictor\_var,outcome\_var)

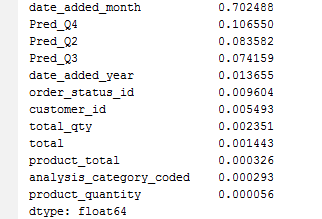
Accuracy : 100.000% Cross-Validation Score : 100.000%

Here we see that the accuracy is 100% for the training set. This is the ultimate case of overfitting and can be resolved in two ways:

1. Reducing the number of predictors
2. Tuning the model parameters

Let’s try both of these. First we see the feature importance matrix from which we’ll take the most important features.

*#Create a series with feature importances:*featimp = pd.Series(model.feature\_importances\_, index=predictor\_var).sort\_values(ascending=**False**)  
print (featimp)



Let’s use the top 5 variables for creating a model and also we are predicting for quarter, its better to remove the month, it will have some bias. Also, we will modify the parameters of random forest model a little bit:

*# after removing correlated variable*model = RandomForestClassifier(n\_estimators=25, min\_samples\_split=25, max\_depth=7, max\_features=1)  
predictor\_var = [**'customer\_id'**,**'total'**,**'order\_status\_id'**,**'date\_added\_year'**,**'analysis\_category\_coded'**]  
classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 77.414% Cross-Validation Score : 73.073%

Notice that although accuracy reduced, but the cross-validation score is improving showing that the model is generalizing well. Remember that random forest models are not exactly repeatable. Different runs will result in slight variations because of randomization. But the output should stay in the ballpark.

You would have noticed that even after some basic parameter tuning on random forest, we have reached a cross-validation accuracy only slightly better than the original logistic regression model.

Model Training and Testing:

I have splitted the dataset in to 2 parts one is for training and other is for testing using the cross validation by Kfold. Below is the code snippet. It will randomly do the split to get the best accuracy.

*# Perform k-fold cross-validation with 5 folds*kf = KFold(df.shape[0], n\_folds=5)  
error = []  
**for** train, test **in** kf:  
 *# Filter training data* train\_predictors = (df[predictors].iloc[train, :])  
  
 *# The target we're using to train the algorithm.* train\_target = df[outcome].iloc[train]  
  
 *# Filter training data* test\_predictors = (df[predictors].iloc[test, :])  
  
 *# The target we're using to train the algorithm.* test\_target = df[outcome].iloc[test]

Model Comparision and accuracy:

We would often use a confusion matrix to find out our type I and type II error rate.

A confusion matrix CC is such that Ci,jCi,j is the number of predictions known to be in group ii but predicted to be in group j.

Because the confusion matrix is such A Ci,jCi,j is the number of predictions known to be in group ii but predicted to be in group jj, we have:

We can plot this using an ROC curve, where we plot the True Positive rate against the False Positive rate, in which a large area under the curve is more favourable.

