**Sections**

* ### Exploratory Data Analysis ###
  + [Missing Value](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Missing%20Value%20Treatment)
  + [Univariate Analysis](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Univariate%20Analysis)
    - *Continuous Variables*
    - *Categorical Variables*
  + [Bi-variate Analysis](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Bi-variate%20Analysis)
    - *Categorical & Continuous*
    - *Categorical & Categorical*
  + [Outlier](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Outlier)
* ### Data Preparation ###
  + [Label encoding](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Label%20encoding)
  + [Dummy variables](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Dummy%20Variables)
  + [Spliting the data](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Data%20Spliting)
* ### Machine Learning ###
  + [Regression](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Regression)
  + [Decision Tree](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Decision%20Tree)
  + [Random Forest](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Random%20Forest)
  + [Gredient Boosting with H2O](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Gredient%20Boosting%20with%20H2O)
  + [Deep Learning with H2O](https://render.githubusercontent.com/view/ipynb?commit=3f4b5bc5d02d0f22dcd9716e93da690afae3715c&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f766169626861766c617475726b61722f426c61636b2d4672696461792d50726f626c656d2f336634623562633564303264306632326463643937313665393364613639306166616533373135632f426c61636b25323046726964617925323050726f626c656d2e6970796e62&nwo=vaibhavlaturkar%2FBlack-Friday-Problem&path=Black+Friday+Problem.ipynb&repository_id=70607833&repository_type=Repository#Deep%20Learning%20with%20H2O)

The competition is a regression problem, in which we have to predict how much a person will purchase a product in future when we know his/her purchase history.

This information is important to an organization, as it clearly helps them decode people's preferences and to create personalized offers for customers against different products.

Every row in the dataset represents a user and his/her purchase details. So you have a unique user\_id, a unique product\_id and the total purchase done by the user for that product.

In [1]:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib** **as** **mp**

**import** **seaborn** **as** **sns**

**import** **matplotlib.pyplot** **as** **plt**

%matplotlib inline

**from** **scipy** **import** stats

In [2]:

train=pd.read\_csv("G:/Black Friday/train.csv")

test=pd.read\_csv("G:/Black Friday/test-comb.csv")

test1 = test

test.head()

In [9]:

train.describe()

In [16]:

train.isnull().values.any()

In [18]:

train.dtypes *## Find data types*

In [20]:

train.info()

**Missing Value Treatment**

***Why missing values treatment is required?***

Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

***Why my data has missing values?***

We looked at the importance of treatment of missing values in a dataset. Now, let’s identify the reasons for occurrence of these missing values. They may occur at two stages:

**Data Extraction**: It is possible that there are problems with extraction process. In such cases, we should double-check for correct data with data guardians. Some hashing procedures can also be used to make sure data extraction is correct. Errors at data extraction stage are typically easy to find and can be corrected easily as well.

**Data collection**: These errors occur at time of data collection and are harder to correct. They can be categorized in four types:

1)**Missing completely at random**: This is a case when the probability of missing variable is same for all observations. For example: respondents of data collection process decide that they will declare their earning after tossing a fair coin. If an head occurs, respondent declares his / her earnings & vice versa. Here each observation has equal chance of missing value.

2)**Missing at random**: This is a case when variable is missing at random and missing ratio varies for different values / level of other input variables. For example: We are collecting data for age and female has higher missing value compare to male.

3)**Missing that depends on unobserved predictors**: This is a case when the missing values are not random and are related to the unobserved input variable. For example: In a medical study, if a particular diagnostic causes discomfort, then there is higher chance of drop out from the study. This missing value is not at random unless we have included “discomfort” as an input variable for all patients.

4)**Missing that depends on the missing value itself**: This is a case when the probability of missing value is directly correlated with missing value itself. For example: People with higher or lower income are likely to provide non-response to their earning.

**\*Which are the methods to treat missing values ?**

**Deletion**: It is of two types: List Wise Deletion and Pair Wise Deletion. In list wise deletion, we delete observations where any of the variable is missing. Simplicity is one of the major advantage of this method, but this method reduces the power of model because it reduces the sample size. In pair wise deletion, we perform analysis with all cases in which the variables of interest are present. Advantage of this method is, it keeps as many cases available for analysis. One of the disadvantage of this method, it uses different sample size for different variables.

Deletion methods are used when the nature of missing data is “Missing completely at random” else non random missing values can bias the model output.

**Mean/ Mode/ Median Imputation**: Imputation is a method to fill in the missing values with estimated ones. The objective is to employ known relationships that can be identified in the valid values of the data set to assist in estimating the missing values. Mean / Mode / Median imputation is one of the most frequently used methods. It consists of replacing the missing data for a given attribute by the mean or median (quantitative attribute) or mode (qualitative attribute) of all known values of that variable. It can be of two types:- ***Generalized Imputation***: In this case, we calculate the mean or median for all non missing values of that variable then replace missing value with mean or median. Like in above table, variable “Manpower” is missing so we take average of all non missing values of “Manpower” (28.33) and then replace missing value with it. Similar case Imputation: In this case, we calculate average for gender “Male” (29.75) and “Female” (25) individually of non missing values then replace the missing value based on gender. For “Male“, we will replace missing values of manpower with 29.75 and for “Female” with 25.

**Prediction Model**: Prediction model is one of the sophisticated method for handling missing data. Here, we create a predictive model to estimate values that will substitute the missing data. In this case, we divide our data set into two sets: One set with no missing values for the variable and another one with missing values. First data set become training data set of the model while second data set with missing values is test data set and variable with missing values is treated as target variable. Next, we create a model to predict target variable based on other attributes of the training data set and populate missing values of test data set.We can use regression, ANOVA, Logistic regression and various modeling technique to perform this. There are 2 drawbacks for this approach: 1)The model estimated values are usually more well-behaved than the true values 2)If there are no relationships with attributes in the data set and the attribute with missing values, then the model will not be precise for estimating missing values.

**KNN Imputation**: In this method of imputation, the missing values of an attribute are imputed using the given number of attributes that are most similar to the attribute whose values are missing. The similarity of two attributes is determined using a distance function. It is also known to have certain advantage & disadvantages.

*Advantages*: k-nearest neighbour can predict both qualitative & quantitative attributes Creation of predictive model for each attribute with missing data is not required Attributes with multiple missing values can be easily treated Correlation structure of the data is taken into consideration

*Disadvantage*: KNN algorithm is very time-consuming in analyzing large database. It searches through all the dataset looking for the most similar instances.Choice of k-value is very critical. Higher value of k would include attributes which are significantly different from what we need whereas lower value of k implies missing out of significant attributes.

After dealing with missing values, the next task is to deal with outliers. Often, we tend to neglect outliers while building models. This is a discouraging practice. Outliers tend to make your data skewed and reduces accuracy. Let’s learn more about outlier treatment.

In [20]:

train.isnull().sum() *## Find number of NA in each colunm*

In [69]:

train.isnull().sum().sum() *# Total NA in data*

In [4]:

train['Product\_Category\_2'].median() *## Find median of Product\_Category\_2 to fill NA*

In [3]:

train['Product\_Category\_2'].fillna(value=9,inplace=**True**) *## Filling NA for Product\_Category\_2*

In [71]:

train['Product\_Category\_3'].median() *## Find median of Product\_Category\_2 to fill NA*

In [4]:

train['Product\_Category\_3'].fillna(value=14,inplace=**True**) *## Filling NA for Product\_Category\_2*

In [120]:

train.isnull().sum().sum() *## There are no NA in the train data*

In [16]:

test.isnull().sum() *## Find NA in test data*

In [5]:

test['Product\_Category\_3'].median() *## median to fill the NA in test product category 3 colunm*

In [5]:

test['Product\_Category\_3'].fillna(value=14,inplace=**True**) *## fill NA by median*

In [75]:

test['Product\_Category\_2'].median() *## median to fill the NA in test product category 3 colunm*

In [6]:

test['Product\_Category\_2'].fillna(value=9,inplace=**True**) *## fill NA by median*

In [35]:

test.isnull().sum() *## No NA in test data*

In [55]:

train.head(3)

In [57]:

train.shape

In [7]:

*## Find Unique value in Product ID*

train.User\_ID.nunique()

In [62]:

*## Number of unique in Product ID*

train.Product\_ID.nunique() *## There are 3631 unique product ID outof 550068 observations*

In [65]:

*## Number of Categorical variable*

(train.dtypes=='object').sum() *## There are 5 categorical variables*

In [67]:

*## Find colunm difference in test & train*

set(test.columns).difference(set(train.columns))

**Univariate Analysis**

At this stage, we explore variables one by one. Method to perform uni-variate analysis will depend on whether the variable type is categorical or continuous. Let’s look at these methods and statistical measures for categorical and continuous variables individually:

**Continuous Variables**:- In case of continuous variables, we need to understand the central tendency and spread of the variable. These are measured using various statistical metrics visualization methods like **Histogram & Boxplot**. We will use Matplotlib & Seaborn package.

In [8]:

*## which continuous variables in data*

train.dtypes *## data types of variables*

In [15]:

(train.dtypes=='int64').sum() *# there are 5 variables which have integer as dtype*

In [3]:

(train.dtypes=='float64').sum() *# there are 5 variables which have float as dtype*

In [25]:

*# Check the distribution of dependent variable i.e Purchase*

train['Purchase'].hist(bins=20).set\_title('Purchase Pattern')

Looking at Purchase Pattern we can see that number of count is more at 6000 & 7000 and above 15000 have very less count.

In [9]:

*# Check the distribution of dependent variable i.e Purchase*

train['Product\_Category\_1'].hist(bins=20).set\_title('Product\_Category\_1 Pattern')

In [10]:

*# Check the distribution of Product\_Category\_2*

train['Product\_Category\_2'].hist(bins=20).set\_title('Product\_Category\_2 Pattern')

In [11]:

*# Check the distribution of Product\_Category\_3*

train['Product\_Category\_3'].hist(bins=20).set\_title('Product\_Category\_3 Pattern')

In [12]:

*# Check the distribution of Occupation*

train['Occupation'].hist(bins=20).set\_title('Occupation Pattern')

**Categorical Variables**:- For categorical variables, we’ll use frequency table to understand distribution of each category. We can also read as percentage of values under each category. It can be be measured using two metrics, Count and Count% against each category. Bar chart can be used as visualization.

In [13]:

train.dtypes

In [55]:

*# Number of Gender count*

dim = (11,5)

fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=**True**,figsize=dim)

sns.countplot(x="Gender", data=train, palette="Greens\_d",ax=ax1).set\_title('Train data: Gender-wise count');

sns.countplot(x='Gender',data=test,palette="GnBu\_d",ax=ax2).set\_title('Test data: Gender-wise count')

There are more Males who are purchasing in this data

In [54]:

*## Number of count of Age*

dim = (11,5)

fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=**True**,figsize=dim)

sns.countplot(x='Age',data=train,palette="Blues",ax=ax1).set\_title('Train data: Agewise count');

sns.countplot(x='Age',data=test,palette="BuGn\_r",ax=ax2).set\_title('Test data: Agewise count')

There are many count in Age 26-35 and very less in 0-17 in train & test data

In [53]:

*## Number of count of City*

dim = (11,5)

fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=**True**,figsize=dim)

sns.countplot(x='City\_Category',data=train,palette="Blues",ax=ax1).set\_title('Train data: Citywise count');

sns.countplot(x='City\_Category',data=test,palette="BuGn\_r",ax=ax2).set\_title('Test data: Citywise count')

People from City B are purchasing more

In [56]:

*## Number of count of stay*

dim = (11,5)

fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=**True**,figsize=dim)

sns.countplot(x='Stay\_In\_Current\_City\_Years',data=train,palette="Blues",ax=ax1).set\_title('Train data: Stay-wise count');

sns.countplot(x='Stay\_In\_Current\_City\_Years',data=test,palette="BuGn\_r",ax=ax2).set\_title('Test data: Stay-wise count')

People who have stayed for 1 year are purching more compared to others

In [57]:

*## Number of count of Marital\_Status*

dim = (11,5)

fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=**True**,figsize=dim)

sns.countplot(x='Marital\_Status',data=train,palette="Blues",ax=ax1).set\_title('Train data: Marital\_Status-wise count');

sns.countplot(x='Marital\_Status',data=test,palette="BuGn\_r",ax=ax2).set\_title('Test data: Marital\_Status-wise count')

There are no people who are married in Test data

In [61]:

*## Number of count of Product\_ID*

dim = (11,5)

fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=**True**,figsize=dim)

sns.countplot(x='Product\_ID',data=train,palette="Blues",ax=ax1).set\_title('Train data: Product\_ID-wise count');

sns.countplot(x='Product\_ID',data=test,palette="BuGn\_r",ax=ax2).set\_title('Test data: Product\_ID -wise count')

**Bi-variate Analysis**

Bi-variate Analysis finds out the relationship between two variables. Here, we look for association and disassociation between variables at a pre-defined significance level. We can perform bi-variate analysis for any combination of categorical and continuous variables. The combination can be: Categorical & Categorical, Categorical & Continuous and Continuous & Continuous. Different methods are used to tackle these combinations during analysis process.

Let’s understand the possible combinations in detail:

**Continuous & Continuous**: While doing bi-variate analysis between two continuous variables, we should look at **scatter plot**. It is a nifty way to find out the relationship between two variables. The pattern of scatter plot indicates the relationship between variables. The relationship can be linear or non-linear.

In [62]:

train.dtypes

***There are no Continuous variable in our data set***

**Categorical & Continuous**: While exploring relation between categorical and continuous variables, we can draw box plots for each level of categorical variables. If levels are small in number, it will not show the statistical significance. To look at the statistical significance we can perform Z-test, T-test or ANOVA.

1) *Z-Test/ T-Test*:- Either test assess whether mean of two groups are statistically different from each other or not.

**\*If the probability of Z is small then the difference of two averages is more significant**. The T-test is very similar to Z-test but it is used when number of observation for both categories is less than 30.

2) *ANOVA*:- It assesses whether the average of more than two groups is statistically different.

Example: Suppose, we want to test the effect of five different exercises. For this, we recruit 20 men and assign one type of exercise to 4 men (5 groups). Their weights are recorded after a few weeks. We need to find out whether the effect of these exercises on them is significantly different or not. This can be done by comparing the weights of the 5 groups of 4 men each.

In [7]:

**from** **scipy** **import** stats

In [17]:

gender\_grp = train.groupby(['Gender'])

gender\_grp['Purchase'].describe()

In [35]:

gender\_grp.boxplot(column=['Purchase'],return\_type='axes')

**Boxplot shows Female & Male are not significantly different but we will check by Hypothesis**

In [11]:

*# make purchase data for hypotheis of Female & Male to find out either F & M are significantly different*

M\_purchase = train[train['Gender']=='F']['Purchase']

F\_purchase = train[train['Gender']=='M']['Purchase']

In [16]:

stats.ttest\_ind(M\_purchase, F\_purchase,equal\_var=**False**) *## equal\_var means if True means they have same sd*

**By doing Hypothesis we can say that they have mean purchase difference i.e they are significant**

In [36]:

stats.f\_oneway(M\_purchase, F\_purchase)

In [46]:

*## Age wise hypothesis*

age\_grp = train.groupby(['Age'])

age\_grp.boxplot(column=['Purchase'],return\_type='axes',figsize=(10,10))

In [47]:

age\_grp['Purchase'].describe()

In [67]:

*## subset the data as per levels from Age*

p\_0to17 = train[train['Age']=='0-17']['Purchase']

p\_18to25 = train[train['Age']=='18-25']['Purchase']

p\_26to35 = train[train['Age']=='26-35']['Purchase']

p\_36to45 = train[train['Age']=='36-45']['Purchase']

p\_46to50 = train[train['Age']=='46-50']['Purchase']

p\_51to55 = train[train['Age']=='51-55']['Purchase']

p\_55 = train[train['Age']=='55+']['Purchase']

In [71]:

*## Test the anova*

stats.f\_oneway(p\_0to17,p\_18to25,p\_26to35,p\_36to45,p\_46to50,p\_51to55,p\_55)

*Age groups have different purchase mean,so we will consider this Age variable for model building*

In [75]:

grp\_marital = train.groupby(['Marital\_Status'])

grp\_marital['Purchase'].describe()

In [82]:

grp\_marital.boxplot(column=['Purchase'],return\_type='axes',figsize=(10,5))

In [73]:

*## subset Marital status*

m0 = train[train['Marital\_Status']== 0]['Purchase']

m1 = train[train['Marital\_Status']== 1]['Purchase']

*## Test Anova*

stats.f\_oneway(m0,m1)

*As we show above at Univariate analysis that Marital\_Status = 1 is not present in Test data so we have decided to remove Marital\_Status= 1 from Train data but after doing Bivariate analysis we have found out that Marital Status groups 1 & 0 are not significantly different from one another they have equal purchase mean so while doing regression we will not consider this variable for model building.*

In [83]:

city\_grp = train.groupby(['City\_Category'])

city\_grp['Purchase'].describe()

In [87]:

city\_grp.boxplot(column=['Purchase'],return\_type='axes',figsize=(10,10))

In [89]:

*# Subset City\_Category*

city\_A = train[train['City\_Category']=='A']['Purchase']

city\_B = train[train['City\_Category']=='B']['Purchase']

city\_C = train[train['City\_Category']=='C']['Purchase']

*# Test ANOVA*

stats.f\_oneway(city\_A,city\_B,city\_C)

*City\_Category variables groups have signifiacntly different purchase mean, we can take this variable for analysis.*

In [91]:

*# group Stay\_In\_Current\_City\_Years*

grp\_yrs = train.groupby(['Stay\_In\_Current\_City\_Years'])

grp\_yrs['Purchase'].describe()

In [93]:

grp\_yrs.boxplot(column=['Purchase'],return\_type='axes',figsize=(10,10))

In [100]:

*# subset Stay\_In\_Current\_City\_Years*

yr0 = train[train['Stay\_In\_Current\_City\_Years']=='0']['Purchase']

yr1 = train[train['Stay\_In\_Current\_City\_Years']=='1']['Purchase']

yr2 = train[train['Stay\_In\_Current\_City\_Years']=='2']['Purchase']

yr3 = train[train['Stay\_In\_Current\_City\_Years']=='3']['Purchase']

yr4 = train[train['Stay\_In\_Current\_City\_Years']=='4+']['Purchase']

*# test anova*

stats.f\_oneway(yr0,yr1,yr2,yr3,yr4)

*Stay\_In\_Current\_City\_Years groups have same purchase mean*

In [103]:

*# group Product\_ID*

grp\_pid = train.groupby(['Product\_ID'])

grp\_pid['Purchase'].describe()

In [106]:

*# subset Product ID*

p1 = train[train['Product\_ID']=='P00000142']['Purchase']

p2 = train[train['Product\_ID']=='P00000242']['Purchase']

p3 = train[train['Product\_ID']=='P00000342']['Purchase']

p4 = train[train['Product\_ID']=='P00000442']['Purchase']

p5 = train[train['Product\_ID']=='P0099642']['Purchase']

p6 = train[train['Product\_ID']=='P0099742']['Purchase']

p7 = train[train['Product\_ID']=='P0099842']['Purchase']

p8 = train[train['Product\_ID']=='P0099942']['Purchase']

*# test anova*

stats.f\_oneway(p1,p2,p3,p4,p5,p6,p7,p8)

*Product ID groups have significant difference in purchase mean*

In [112]:

*# group Product\_Category\_1*

grp\_cat1 = train.groupby(['Product\_Category\_1'])

grp\_cat1['Purchase'].describe()

*# group Product\_Category\_2*

grp\_cat2 = train.groupby(['Product\_Category\_2'])

grp\_cat2['Purchase'].describe()

In [116]:

*# group Product\_Category\_3*

grp\_cat3 = train.groupby(['Product\_Category\_3'])

grp\_cat3['Purchase'].describe()

In [117]:

*# group Occupation*

grp\_occ = train.groupby(['Occupation'])

grp\_occ['Purchase'].describe()

In [118]:

train.dtypes

In [125]:

*### Pivot Table*

impute\_grps = train.pivot\_table(values=["Purchase"], index=["Gender","City\_Category","Stay\_In\_Current\_City\_Years","Occupation"], aggfunc=np.mean)

**Categorical & Categorical**: To find the relationship between two categorical variables, we can use following methods:

*Two-way table*: We can start analyzing the relationship by creating a two-way table of count and count%. The rows represents the category of one variable and the columns represent the categories of the other variable. We show count or count% of observations available in each combination of row and column categories.

*Stacked Column Chart*: This method is more of a visual form of Two-way table.

*Chi-Square Test*: This test is used to derive the statistical significance of relationship between the variables. Also, it tests whether the evidence in the sample is strong enough to generalize that the relationship for a larger population as well. Chi-square is based on the difference between the expected and observed frequencies in one or more categories in the two-way table. It returns probability for the computed chi-square distribution with the degree of freedom.

Probability of 0: It indicates that both categorical variable are dependent Probability of 1: It shows that both variables are independent. Probability less than 0.05: It indicates that the relationship between the variables is significant at 95% confidence.

In [10]:

train.dtypes

In [29]:

*## Stacked chart for visualization*

var = train.groupby(['Age','Gender']).Gender.count()

var.unstack().plot(kind='bar',stacked=**True**, grid=**False**,figsize=(10,5))

return np.sum(name == np.asarray(self.names)) > 1

In [21]:

*## Chi-sq test*

chi\_AG = pd.crosstab(train['Gender'],train['Age'])

chi\_AG

In [30]:

stats.chi2\_contingency(chi\_AG)

In [52]:

*# stack*

var1 = train.groupby(['City\_Category','Gender']).Gender.count()

var1.unstack().plot(kind='bar',stacked=**True**, grid=**False**,figsize=(10,5),color=['pink','lightgreen'])

In [33]:

*## cross tab*

chi\_GC = pd.crosstab(train['Gender'],train['City\_Category'])

chi\_GC

In [34]:

stats.chi2\_contingency(chi\_GC)

In [48]:

*# stack*

var2 = train.groupby(['Marital\_Status','Gender']).Gender.count()

var2.unstack().plot(kind='bar',stacked=**True**, grid=**False**,figsize=(10,5),color=['grey','lightblue'])

In [36]:

*## cross tab*

chi\_Gm = pd.crosstab(train['Gender'],train['Marital\_Status'])

In [37]:

stats.chi2\_contingency(chi\_Gm)

**Outlier**

Outlier is a commonly used terminology by analysts and data scientists as it needs close attention else it can result in wildly wrong estimations. Simply speaking, Outlier is an observation that appears far away and diverges from an overall pattern in a sample.

**What are the types of Outliers?**

Outlier can be of two types: Univariate and Multivariate. Above, we have discussed the example of univariate outlier. These outliers can be found when we look at distribution of a single variable. Multi-variate outliers are outliers in an n-dimensional space. In order to find them, you have to look at distributions in multi-dimensions.

In [65]:

plt.figure(figsize=(15, 4))

sns.boxplot(train['Purchase'],color='c')

**Variable Transformation**

What are the common methods of Variable Transformation?

There are various methods used to transform variables. As discussed, some of them include square root, cube root, logarithmic, binning, reciprocal and many others. Let’s look at these methods in detail by highlighting the pros and cons of these transformation methods.

**Logarithm**: Log of a variable is a common transformation method used to change the shape of distribution of the variable on a distribution plot. It is generally used for reducing right skewness of variables. Though, It can’t be applied to zero or negative values as well.

**Square / Cube root**: The square and cube root of a variable has a sound effect on variable distribution. However, it is not as significant as logarithmic transformation. Cube root has its own advantage. It can be applied to negative values including zero. Square root can be applied to positive values including zero.

**Binning**: It is used to categorize variables. It is performed on original values, percentile or frequency. Decision of categorization technique is based on business understanding. For example, we can categorize income in three categories, namely: High, Average and Low. We can also perform co-variate binning which depends on the value of more than one variables.

In [46]:

plt.figure(figsize=(10, 6))

sns.distplot(train['Purchase'])

**Data Preparation**

**Label Encoding**

In [7]:

**from** **sklearn.preprocessing** **import** LabelEncoder

le=LabelEncoder()

data=train['User\_ID'].append(test['User\_ID'])

le.fit(data.values)

train['User\_ID']=le.transform(train['User\_ID'])

test['User\_ID']=le.transform(test['User\_ID'])

In [8]:

**from** **sklearn.preprocessing** **import** LabelEncoder

le=LabelEncoder()

data=train['Occupation'].append(test['Occupation'])

le.fit(data.values)

train['Occupation']=le.transform(train['Occupation'])

test['Occupation']=le.transform(test['Occupation'])

In [9]:

*#Label Encoding*

train\_x = train.ix[:,[0,1,2,3,4,5,6,7,8,9,10,11]] *#for subseting by index*

test\_x = test.ix[:,[1,2,3,4,5,6,7,8,9,10,11]]

*#Importing LabelEncoder and initializing it*

**from** **sklearn.preprocessing** **import** LabelEncoder

le=LabelEncoder()

**for** col **in** train\_x.columns.values:

*# Encoding only categorical variables*

**if** train\_x[col].dtypes=='object':

*# Using whole data to form an exhaustive list of levels*

data=train\_x[col].append(test\_x[col])

le.fit(data.values)

train\_x[col]=le.transform(train\_x[col])

test\_x[col]=le.transform(test\_x[col])

C:\Users\Thoshiba\AppData\Roaming\Python\Python35\site-packages\ipykernel\\_\_main\_\_.py:17: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy

**Dummy Variables**

In [12]:

*## Create dummy variables*

train\_x = pd.get\_dummies(train\_x, columns=['City\_Category','Age','Stay\_In\_Current\_City\_Years'])

test\_x = pd.get\_dummies(test\_x, columns=['City\_Category','Age','Stay\_In\_Current\_City\_Years'])

print (train\_x.shape)

print (test\_x.shape)

(550068, 24)

(233599, 23)

**Data Spliting**

In [10]:

*### Convert User ID & Product\_ID in*

train['User\_ID'] = train['User\_ID'].astype('category')

test['User\_ID'] = test['User\_ID'].astype('category')

train['Product\_ID'] = train['Product\_ID'].astype('category')

train['User\_ID'] = test['Product\_ID'].astype('category')

In [48]:

*#from sklearn.cross\_validation import train\_test\_split*

*#x\_train,x\_test = train\_test\_split(train,test\_size=0.30)*

In [11]:

*## Split the data for to check the accuracy*

**from** **sklearn.cross\_validation** **import** train\_test\_split

x\_train,x\_test = train\_test\_split(train\_x,test\_size=0.30)

In [12]:

print (x\_train.shape)

print (x\_test.shape)

(385047, 12)

(165021, 12)

In [13]:

x\_test.head()

**Linear Regression**

* Without Dummy Variables
* With Dummy Variables

In [24]:

**import** **statsmodels.formula.api** **as** **smf**

*# create a fitted model in one line*

lm = smf.ols(formula='Purchase ~Age\_0+Age\_1+Age\_2+Age\_3+Age\_4+Age\_5+Age\_6+Gender+Occupation+Product\_ID+City\_Category\_0+City\_Category\_1+City\_Category\_2+City\_Category\_0+Product\_Category\_1+Stay\_In\_Current\_City\_Years\_0+Stay\_In\_Current\_City\_Years\_1+Stay\_In\_Current\_City\_Years\_2+Stay\_In\_Current\_City\_Years\_3+Stay\_In\_Current\_City\_Years\_4+Product\_Category\_2+Product\_Category\_3', data=x\_train).fit()

*# print the coefficients*

print (lm.summary())

OLS Regression Results

==============================================================================

Dep. Variable: Purchase R-squared: 0.134

Model: OLS Adj. R-squared: 0.134

Method: Least Squares F-statistic: 3305.

Date: Thu, 13 Oct 2016 Prob (F-statistic): 0.00

Time: 17:07:31 Log-Likelihood: -3.8002e+06

No. Observations: 385047 AIC: 7.601e+06

Df Residuals: 385028 BIC: 7.601e+06

Df Model: 18

Covariance Type: nonrobust

================================================================================================

coef std err t P>|t| [95.0% Conf. Int.]

------------------------------------------------------------------------------------------------

Intercept 7436.5852 29.128 255.310 0.000 7379.496 7493.675

Age\_0 553.5046 40.425 13.692 0.000 474.273 632.736

Age\_1 833.0344 18.651 44.665 0.000 796.479 869.589

Age\_2 1041.1772 15.030 69.274 0.000 1011.719 1070.635

Age\_3 1170.5236 18.133 64.551 0.000 1134.983 1206.064

Age\_4 1125.8243 25.026 44.986 0.000 1076.774 1174.875

Age\_5 1459.3213 26.909 54.231 0.000 1406.580 1512.063

Age\_6 1253.1997 34.708 36.107 0.000 1185.174 1321.226

Gender 533.4961 17.674 30.186 0.000 498.856 568.136

Occupation 6.6379 1.174 5.653 0.000 4.336 8.939

Product\_ID -0.4210 0.007 -56.956 0.000 -0.436 -0.407

City\_Category\_0 2181.4001 15.432 141.360 0.000 2151.155 2211.645

City\_Category\_1 2352.6261 13.942 168.748 0.000 2325.301 2379.951

City\_Category\_2 2902.5590 14.566 199.274 0.000 2874.011 2931.107

Product\_Category\_1 -406.7163 2.048 -198.586 0.000 -410.730 -402.702

Stay\_In\_Current\_City\_Years\_0 1453.2916 18.852 77.088 0.000 1416.342 1490.242

Stay\_In\_Current\_City\_Years\_1 1480.8574 13.647 108.510 0.000 1454.109 1507.606

Stay\_In\_Current\_City\_Years\_2 1520.0992 16.746 90.775 0.000 1487.278 1552.920

Stay\_In\_Current\_City\_Years\_3 1493.8881 17.234 86.680 0.000 1460.109 1527.667

Stay\_In\_Current\_City\_Years\_4 1488.4489 18.021 82.595 0.000 1453.128 1523.770

Product\_Category\_2 -58.6110 2.047 -28.635 0.000 -62.623 -54.599

Product\_Category\_3 -12.2218 3.542 -3.451 0.001 -19.164 -5.280

==============================================================================

Omnibus: 41270.198 Durbin-Watson: 1.999

Prob(Omnibus): 0.000 Jarque-Bera (JB): 57288.648

Skew: 0.858 Prob(JB): 0.00

Kurtosis: 3.793 Cond. No. 1.83e+17

==============================================================================

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 4.64e-23. This might indicate that there are

strong multicollinearity problems or that the design matrix is singular.

In [25]:

*## Accuracy for full model*

predict = lm.predict(x\_test)

MSE = np.mean((x\_test['Purchase'] - predict) \*\* 2)

print("calculate RMSE: **%.2f**" %np.sqrt(MSE))

calculate RMSE: 4660.09

we will remove ***Stay\_In\_Current\_City\_Years*** variable because it is having p-value more than 0.05 which says it is not a significant variable in our model.

In [36]:

*## Removing Stay\_In\_Current\_City\_Years & Marital Status*

*# create a fitted model in one line*

lm1 = smf.ols(formula='Purchase ~Age\_0+Age\_1+Age\_2+Age\_3+Age\_4+Age\_5+Age\_6+Gender+Occupation+Product\_ID+City\_Category\_0+City\_Category\_1+City\_Category\_0+Product\_Category\_1+Product\_Category\_2+Product\_Category\_3', data=x\_train).fit()

*# print the coefficients*

print (lm1.summary())

OLS Regression Results

==============================================================================

Dep. Variable: Purchase R-squared: 0.134

Model: OLS Adj. R-squared: 0.134

Method: Least Squares F-statistic: 4248.

Date: Thu, 13 Oct 2016 Prob (F-statistic): 0.00

Time: 17:42:04 Log-Likelihood: -3.8002e+06

No. Observations: 385047 AIC: 7.601e+06

Df Residuals: 385032 BIC: 7.601e+06

Df Model: 14

Covariance Type: nonrobust

======================================================================================

coef std err t P>|t| [95.0% Conf. Int.]

--------------------------------------------------------------------------------------

Intercept 1.128e+04 43.462 259.504 0.000 1.12e+04 1.14e+04

Age\_0 1103.6859 40.738 27.092 0.000 1023.841 1183.531

Age\_1 1381.2471 19.097 72.329 0.000 1343.818 1418.676

Age\_2 1590.5115 15.658 101.576 0.000 1559.822 1621.201

Age\_3 1720.2880 18.622 92.381 0.000 1683.790 1756.786

Age\_4 1673.3284 25.411 65.849 0.000 1623.523 1723.134

Age\_5 2007.1734 27.290 73.550 0.000 1953.686 2060.661

Age\_6 1802.3966 35.034 51.447 0.000 1733.731 1871.063

Gender 533.3600 17.658 30.205 0.000 498.751 567.969

Occupation 6.6678 1.173 5.682 0.000 4.368 8.968

Product\_ID -0.4210 0.007 -56.955 0.000 -0.435 -0.407

City\_Category\_0 -722.6606 20.214 -35.751 0.000 -762.279 -683.043

City\_Category\_1 -550.1478 17.999 -30.565 0.000 -585.426 -514.870

Product\_Category\_1 -406.7511 2.048 -198.619 0.000 -410.765 -402.737

Product\_Category\_2 -58.6189 2.047 -28.639 0.000 -62.631 -54.607

Product\_Category\_3 -12.2271 3.542 -3.452 0.001 -19.169 -5.285

==============================================================================

Omnibus: 41268.017 Durbin-Watson: 1.999

Prob(Omnibus): 0.000 Jarque-Bera (JB): 57283.279

Skew: 0.858 Prob(JB): 0.00

Kurtosis: 3.793 Cond. No. 8.52e+15

==============================================================================

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 2.15e-20. This might indicate that there are

strong multicollinearity problems or that the design matrix is singular.

In [37]:

*## Accuracy for model without Stay\_In\_Current\_City\_Years*

predict0=lm1.predict(x\_train)

predict1 = lm1.predict(x\_test)

MSE1 = np.mean((x\_test['Purchase'] - predict1) \*\* 2)

print("calculate RMSE: **%.2f**" %np.sqrt(MSE1))

*## No change in RMSE*

calculate RMSE: 4660.12

**High Leverage Plot**

In [38]:

**from** **statsmodels.graphics.regressionplots** **import** plot\_leverage\_resid2

fig, ax = plt.subplots(figsize=(8,6))

fig = plot\_leverage\_resid2(lm1, ax = ax)

**Residual Plot**

In [31]:

*# Scatter plot the training data*

train = plt.scatter(predict0,(x\_train['Purchase']-predict0),c='b',alpha=0.5)

*# Scatter plot the testing data*

test = plt.scatter(predict1,(x\_test['Purchase']-predict1),c='r',alpha=0.5)

*# Plot a horizontal axis line at 0*

plt.hlines(y=0,xmin=-10,xmax=50)

*#Labels*

plt.legend((train,test),('Training','Test'),loc='lower left')

plt.title('Residual Plots')

We will bulid our model on full train and predict on test

In [172]:

train\_x.head()

In [217]:

**import** **statsmodels.formula.api** **as** **smf**

In [82]:

*# create X and y*

feature\_cols = ['Age','Gender','Occupation','Product\_ID','City\_Category','Product\_Category\_1','Product\_Category\_2','Product\_Category\_3']

X = train\_x[feature\_cols]

y = train\_x['Purchase']

*# follow the usual sklearn pattern: import, instantiate, fit*

**from** **sklearn.linear\_model** **import** LinearRegression

lm = LinearRegression()

lm.fit(X, y)

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=False)

In [86]:

*## Take only the feature used in model buliding*

test\_xp = test\_x[feature\_cols]

In [89]:

*# Predict on test*

prediction = lm.predict(test\_xp)

*# Save into DataFrame*

output = pd.DataFrame( data={"User\_ID":test["User\_ID"], "Product\_ID":test['Product\_ID'],"Purchase":prediction} )

*# Use pandas to write the comma-separated output file*

output.to\_csv( "G:/Black Friday/Reg\_Model1.csv", index=**False**,)

We had removed Stay\_years & Marital status variable from the model we got **RMSE: 4671.76** for training model and for complete model we got **RMSE: 4689.54**.   
We will use different alogrithm to improve our accuracy.

In [40]:

train\_x.shape

**Model with Dummy variables**

In [10]:

**import** **statsmodels.formula.api** **as** **smf**

*# create a fitted model in one line*

lm\_full = smf.ols(formula='Purchase ~Age\_0+Age\_1+Age\_2+Age\_3+Age\_4+Age\_5+Age\_6+Gender+Occupation+Product\_ID+City\_Category\_0+City\_Category\_1+City\_Category\_0+Product\_Category\_1+Product\_Category\_2+Product\_Category\_3', data=train\_x).fit()

*# print the coefficients*

print (lm\_full.summary())

*# Predict on test*

prediction = lm\_full.predict(test\_x)

*# Save into DataFrame*

output = pd.DataFrame( data={"User\_ID":test["User\_ID"], "Product\_ID":test['Product\_ID'],"Purchase":prediction} )

*# Use pandas to write the comma-separated output file*

output.to\_csv( "G:/Black Friday/Reg\_Model\_dummy.csv", index=**False**,)

OLS Regression Results

==============================================================================

Dep. Variable: Purchase R-squared: 0.135

Model: OLS Adj. R-squared: 0.135

Method: Least Squares F-statistic: 6116.

Date: Thu, 13 Oct 2016 Prob (F-statistic): 0.00

Time: 20:02:07 Log-Likelihood: -5.4283e+06

No. Observations: 550068 AIC: 1.086e+07

Df Residuals: 550053 BIC: 1.086e+07

Df Model: 14

Covariance Type: nonrobust

======================================================================================

coef std err t P>|t| [95.0% Conf. Int.]

--------------------------------------------------------------------------------------

Intercept 1.132e+04 36.268 312.062 0.000 1.12e+04 1.14e+04

Age\_0 1070.5136 34.134 31.362 0.000 1003.613 1137.415

Age\_1 1396.7311 15.963 87.497 0.000 1365.444 1428.018

Age\_2 1600.2176 13.093 122.217 0.000 1574.555 1625.880

Age\_3 1720.5654 15.546 110.674 0.000 1690.095 1751.036

Age\_4 1689.1928 21.198 79.685 0.000 1647.645 1730.741

Age\_5 2018.7200 22.801 88.535 0.000 1974.030 2063.410

Age\_6 1821.8552 29.363 62.047 0.000 1764.305 1879.405

Gender 527.5609 14.754 35.756 0.000 498.643 556.479

Occupation 6.7829 0.980 6.920 0.000 4.862 8.704

Product\_ID -0.4262 0.006 -68.950 0.000 -0.438 -0.414

City\_Category\_0 -718.7712 16.893 -42.548 0.000 -751.882 -685.661

City\_Category\_1 -551.1833 15.045 -36.636 0.000 -580.671 -521.696

Product\_Category\_1 -406.7224 1.710 -237.911 0.000 -410.073 -403.372

Product\_Category\_2 -57.7835 1.711 -33.777 0.000 -61.136 -54.430

Product\_Category\_3 -15.4652 2.957 -5.229 0.000 -21.262 -9.669

==============================================================================

Omnibus: 58983.268 Durbin-Watson: 1.705

Prob(Omnibus): 0.000 Jarque-Bera (JB): 81912.940

Skew: 0.858 Prob(JB): 0.00

Kurtosis: 3.795 Cond. No. 6.92e+15

==============================================================================

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 4.64e-20. This might indicate that there are

strong multicollinearity problems or that the design matrix is singular.

We got **RMSE 4687. 54**. There was a decrease of 2 with using dummy variables in model.

**Decision Tree Regression**

In [48]:

*## Data Preparation for tree*

feature = ['Age\_0','Age\_1','Age\_2','Age\_3','Age\_4','Age\_5','Age\_6','Gender','Occupation','Product\_ID','City\_Category\_0','City\_Category\_1','City\_Category\_0','Product\_Category\_1','Product\_Category\_2','Product\_Category\_3']

x = x\_train[feature]

y = x\_train['Purchase']

testx = x\_test[feature]

testy = x\_test['Purchase']

*## full data subsetting*

x\_main = train\_x[feature]

y\_main = train\_x['Purchase']

xt\_main = test\_x[feature]

In [27]:

**from** **sklearn** **import** tree

*## Model fitting*

tree = tree.DecisionTreeRegressor(max\_depth = 7)

model\_tree = tree.fit(x, y)

*## Predict*

predict\_tree = model\_tree.predict(testx)

*## Evalute*

MSE = np.mean((testy - predict\_tree) \*\* 2)

print("calculate RMSE: **%.2f**" %np.sqrt(MSE))

calculate RMSE: 2886.41

RMSE reduce from **4460** to **3841** and when we used Depth = 7 then RMSE was 2886.41

**Random Forest Regression**

In [47]:

**from** **sklearn.ensemble** **import** RandomForestRegressor

*## Model fitting*

tree\_rf = RandomForestRegressor(max\_depth=7,n\_estimators=100,verbose = 1)

model\_tree\_rf = tree\_rf.fit(x,y)

*## Predict*

predict\_tree = model\_tree\_rf.predict(testx)

*## Evalute*

MSE = np.mean((testy - predict\_tree) \*\* 2)

print("calculate RMSE: **%.2f**" %np.sqrt(MSE))

After using Random Forest RMSE was further reduced to **2878.08** on validation set

In [51]:

tree\_rf = RandomForestRegressor(max\_depth=7,n\_estimators=200,verbose = 1)

*# Full model fitting with Random forest*

model\_rffull = tree\_rf.fit(x\_main,y\_main)

*# Predict*

prediction = model\_rffull.predict(xt\_main)

*## submission*

*# Save into DataFrame*

output = pd.DataFrame( data={"User\_ID":test["User\_ID"], "Product\_ID":test['Product\_ID'],"Purchase":prediction} )

*# Use pandas to write the comma-separated output file*

output.to\_csv( "G:/Black Friday/RandF\_Model\_dummy200.csv", index=**False**,)

* Got RMSE: **2897.11**  with *100 trees* and *depth 7* .
* Got RMSE: **2897.05**  with *200 trees* and *depth 7* .

In [15]:

*## Data Preparation for tree*

feature = ['Age\_0','Age\_1','Age\_2','Age\_3','Age\_4','Age\_5','Age\_6','Gender','User\_ID','Product\_ID','City\_Category\_0','City\_Category\_1','City\_Category\_0','Product\_Category\_1','Product\_Category\_2','Product\_Category\_3']

x = x\_train[feature]

y = x\_train['Purchase']

testx = x\_test[feature]

testy = x\_test['Purchase']

*## full data subsetting*

x\_main = train\_x[feature]

y\_main = train\_x['Purchase']

xt\_main = test\_x[feature]

In [16]:

**from** **sklearn.ensemble** **import** RandomForestRegressor

*## Model fitting*

tree\_rf = RandomForestRegressor(max\_depth=10,n\_estimators=100,verbose = 1)

model\_tree\_rf = tree\_rf.fit(x,y)

*## Predict*

predict\_tree = model\_tree\_rf.predict(testx)

*## Evalute*

MSE = np.mean((testy - predict\_tree) \*\* 2)

print("calculate RMSE: **%.2f**" %np.sqrt(MSE))

calculated RMSE: 2791.86

In [17]:

**from** **sklearn.ensemble** **import** RandomForestRegressor

*## Model fitting*

tree\_rf = RandomForestRegressor(max\_depth=10,n\_estimators=100,verbose = 1)

*# Full model fitting with Random forest*

model\_rffull = tree\_rf.fit(x\_main,y\_main)

*# Predict*

prediction = model\_rffull.predict(xt\_main)

*## submission*

*# Save into DataFrame*

output = pd.DataFrame( data={"User\_ID":test1["User\_ID"], "Product\_ID":test1['Product\_ID'],"Purchase":prediction} )

*# Use pandas to write the comma-separated output file*

output.to\_csv( "G:/Black Friday/RandForest\_Model.csv", index=**False**,)

**RMSE : 2809.8093**

**Gredient Boosting with H2O**

In [14]:

*## We will import h2o and initialize it*

**import** **h2o**

In [16]:

h2o.init(nthreads=-1)

**from** **h2o.estimators.gbm** **import** H2OGradientBoostingEstimator

Checking whether there is an H2O instance running at http://localhost:54321. connected.

In [17]:

*## Data Preparation for h2o*

*#feature = ['Age\_0','Age\_1','Age\_2','Age\_3','Age\_4','Age\_5','Age\_6','Gender','User\_ID','Product\_ID','City\_Category\_0','City\_Category\_1','Product\_Category\_1','Product\_Category\_2','Product\_Category\_3']*

feature =['User\_ID','Product\_ID']

x = x\_train

x\_train\_h2o = h2o.H2OFrame(x)

testx = x\_test

x\_test\_h2o = h2o.H2OFrame(testx)

In [50]:

*## GBM model fit*

gbm\_regressor = H2OGradientBoostingEstimator(distribution="gaussian",ntrees=500, max\_depth=3,learn\_rate=0.04,nbins\_cats = 5891)

model\_gbm = gbm\_regressor.train(x=feature, y='Purchase',training\_frame=x\_train\_h2o)

print (gbm\_regressor) *## check the model details*

Model Details

=============

H2OGradientBoostingEstimator : Gradient Boosting Machine

Model Key: GBM\_model\_python\_1476449751808\_7

Model Summary:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **number\_of\_trees** | **number\_of\_internal\_trees** | **model\_size\_in\_bytes** | **min\_depth** | **max\_depth** | **mean\_depth** | **min\_leaves** | **max\_leaves** | **mean\_leaves** |
|  | 500.0 | 500.0 | 821648.0 | 0.0 | 3.0 | 1.524 | 1.0 | 8.0 | 4.534 |

ModelMetricsRegression: gbm

\*\* Reported on train data. \*\*

MSE: 6596965.607527898

RMSE: 2568.455880004151

MAE: 1892.381958860219

RMSLE: 0.3400755659183268

Mean Residual Deviance: 6596965.607527898

See the whole table with table.as\_data\_frame()

In [51]:

*# Prediction*

predict = gbm\_regressor.predict(x\_test\_h2o)

*# Copy predictions from H2O to Python*

pred = predict.as\_data\_frame()

*## Evalute*

MSE = np.mean((x\_test['Purchase'] - pred['predict']) \*\* 2)

print("calculate RMSE: **%.2f**" %np.sqrt(MSE))

calculated RMSE: 6562.28

**Complete Fit Model**

In [ ]:

*## Data preparation*

train\_h2o = h2o.H2OFrame(train)

test\_h2o = h2o.H2OFrame(train)

*# Parameters*

gbm\_regressor = H2OGradientBoostingEstimator(distribution="gaussian",ntrees=500, max\_depth=3,learn\_rate=0.04,nbins\_cats = 5891)

*# Model fit*

model\_gbm = gbm\_regressor.train(x=feature, y='Purchase',training\_frame=train\_h2o)

*# Model details*

print (gbm\_regressor)

*# Prediction*

predict = gbm\_regressor.predict(test\_h2o)

*# Copy predictions from H2O to Python*

pred = predict.as\_data\_frame()

*# Make submission Dataframe*

output = pd.DataFrame( data={"User\_ID":test1["User\_ID"], "Product\_ID":test1['Product\_ID'],"Purchase":pred} )

*# Use pandas to write the comma-separated output file*

output.to\_csv( "G:/Black Friday/GBM\_Model.csv", index=**False**,)

**Deep Learning**

In [22]:

**from** **h2o.estimators.deeplearning** **import** H2ODeepLearningEstimator

In [38]:

h20\_deeplearning = H2ODeepLearningEstimator(activation = 'Rectifier',epochs=60,adaptive\_rate =**False**)

In [32]:

feat=x\_train\_h2o.names[1:11]

In [ ]:

model\_deep\_learning = h20\_deeplearning.train(x=feat, y='Purchase',training\_frame=x\_train\_h2o)

In [ ]: