I will do a case study on Computer Prices Prediction Set Obtained from Kaggle. The data contains information about 6259 computer prices based on its features. The goal is to create a predictive model which will predict the Price of the computer.

The flow of the case study is as below :

- Reading the data in python
- Defining the problem statement
- Identifying the Target variable
- Looking at the distribution of Target variable
- Basic Data exploration
- Rejecting useless columns
- Visual Exploratory Data Analysis for data distribution (Histogram and Barcharts)
- Feature Selection based on data distribution
- Outlier treatment
- Missing Values treatment
- Visual correlation analysis
- Statistical correlation analysis (Feature Selection)
- Converting data to numeric for ML
- Sampling and K-fold cross validation
- Trying multiple classification algorithms
- Selecting the best Model
- Deploying the best model in production

Reading the data in python and loading all the libraries

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import numpy as np
```

In [3]:
```
ComputerPriceData=pd.read_csv("C:/Users/RITWIK/OneDrive/Desktop/IVY/Ivy ML/ALL Python ML
```

In [4]:
```
ComputerPriceData.head(5)
```

Out[4]:

	price	speed	hd	ram	screen	cd	multi	premium	ads	trend
0	1499	25	80	4	14	no	no	yes	94	1
1	1795	33	85	2	14	no	no	yes	94	1
2	1595	25	170	4	15	no	no	yes	94	1
3	1849	25	170	8	14	no	no	no	94	1
4	3295	33	340	16	14	no	no	yes	94	1

In [5]:
```
# Shape After Deleting Duplicates
```

```
ComputerPriceData = ComputerPriceData.drop_duplicates()
ComputerPriceData.shape
```

Out[5]:
```
(6183, 10)
```

In [6]:
```
# Finding Categorical and Numerical Data
cat = ComputerPriceData.dtypes[ComputerPriceData.dtypes=='object'].index
num = ComputerPriceData.dtypes[ComputerPriceData.dtypes!='object'].index
```

In [7]:
```
ComputerPriceData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6183 entries, 0 to 6258
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   price    6183 non-null   int64 
 1   speed    6183 non-null   int64 
 2   hd       6183 non-null   int64 
 3   ram      6183 non-null   int64 
 4   screen   6183 non-null   int64 
 5   cd       6183 non-null   object
 6   multi    6183 non-null   object
 7   premium  6183 non-null   object
 8   ads      6183 non-null   int64 
 9   trend    6183 non-null   int64 
dtypes: int64(7), object(3)
memory usage: 531.4+ KB
```

In [8]:
```
ComputerPriceData.nunique()
```

Out[8]:
```
price      808
speed        6
hd          59
ram          6
screen       3
cd           2
multi        2
premium      2
ads         34
trend       35
dtype: int64
```

By looking at the Data we can say that

- **price**: Continuous. Selected.This is the **Target Variable!**
- **speed**: Continuous. Selected
- **hd**: Continuous. Selected
- **ram**: Categorical. Selected
- **screen**: Categorical. Selected
- **cd**: Categorical. Selected
- **multi**: Categorical. Selected
- **premium**: Categorical. Selected
- **ads**: Continuous. Selected
- **trend**: Continuous. Selected

In [9]:
```
ComputerPriceData.columns
```

Out[9]:
```
Index(['price', 'speed', 'hd', 'ram', 'screen', 'cd', 'multi', 'premium',
       'ads', 'trend'],
```

```
       dtype='object')
```

In [10]:
```
# We will convert speed, ram and screen into categorical type

ComputerPriceData["speed"]=ComputerPriceData["speed"].astype("object")
ComputerPriceData["ram"]= ComputerPriceData["ram"].astype("object")
ComputerPriceData["screen"]= ComputerPriceData["screen"].astype("object")
```

In [11]:
```
ComputerPriceData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6183 entries, 0 to 6258
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   price    6183 non-null   int64
 1   speed    6183 non-null   object
 2   hd       6183 non-null   int64
 3   ram      6183 non-null   object
 4   screen   6183 non-null   object
 5   cd       6183 non-null   object
 6   multi    6183 non-null   object
 7   premium  6183 non-null   object
 8   ads      6183 non-null   int64
 9   trend    6183 non-null   int64
dtypes: int64(4), object(6)
memory usage: 531.4+ KB
```

Defining Problem Statement

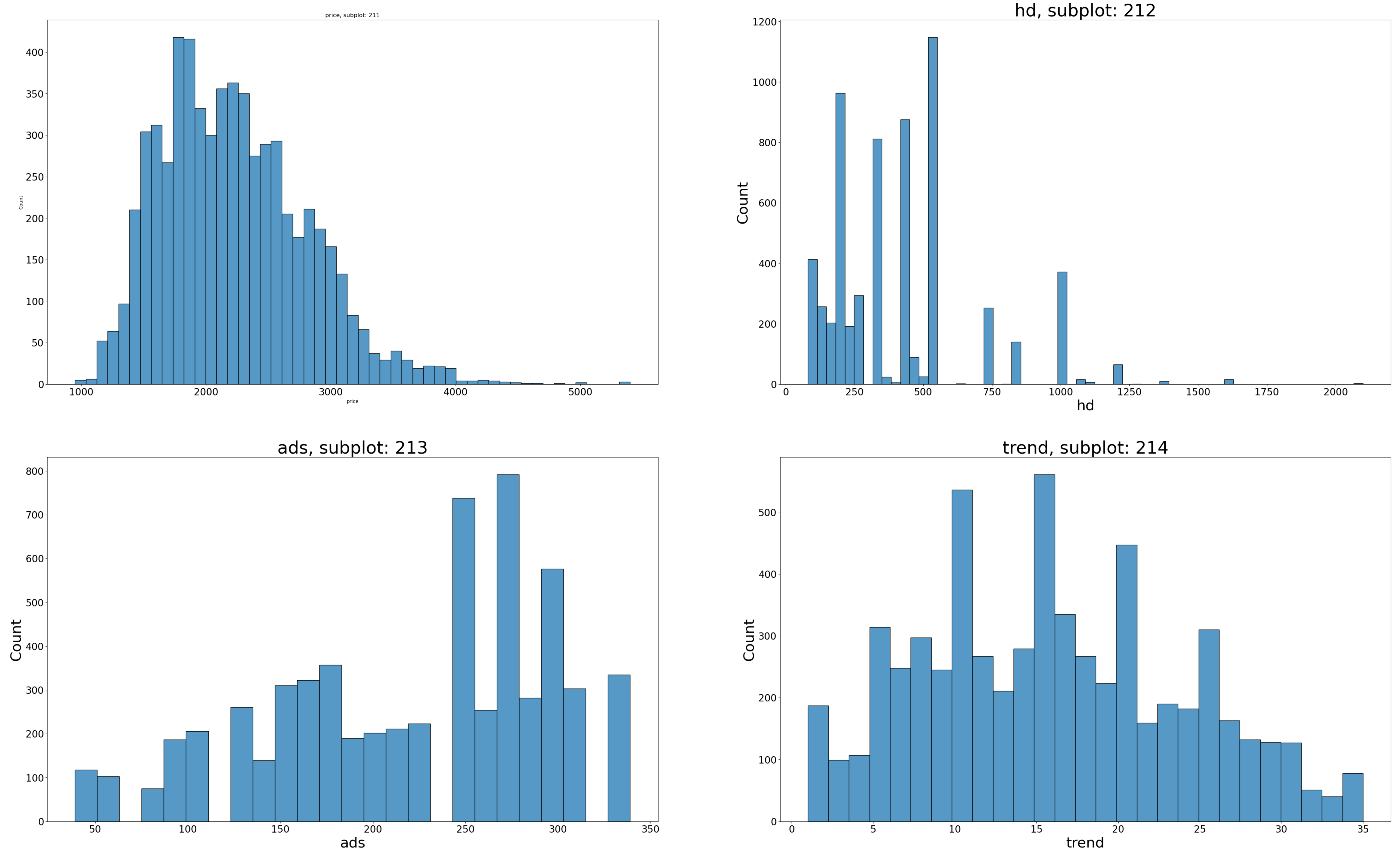
Predicting Price of Computer

- **Target Variable** : Price
- **Predictors**: Ram, Hd, Speed, Screen etc

Visualize distribution of all the Continuous Predictor variables in the data using histograms

In [12]:
```
# num contains all the numerical variables
num = ['price', 'hd', 'ads', 'trend']
```

In [13]:
```
#bivariate plot-political.knowledge using Seaborn lib
fig = plt.figure(figsize=(50,30))
c = 1
for i in num:
    plt.subplot(2, 2, c)
    plt.title('{}, subplot: {}{}{}'.format(i,2,1 ,c))
    plt.xlabel(i)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    #sns.barplot(el_df[i],el_df['age'])
    sns.histplot(x= ComputerPriceData[i], data=ComputerPriceData)
    plt.rcParams.update({'font.size': 30})
    c = c + 1

plt.show()
```



We can notice present of outliers in Price and hd columns

For Categorical columns we will use Bar Charts

Bar Charts Interpretation

These bar charts represent the frequencies of each category in the Y-axis and the category names in the X-axis.

In the ideal bar chart each category has comparable frequency. Hence, there are enough rows for each category in the data for the ML algorithm to learn.

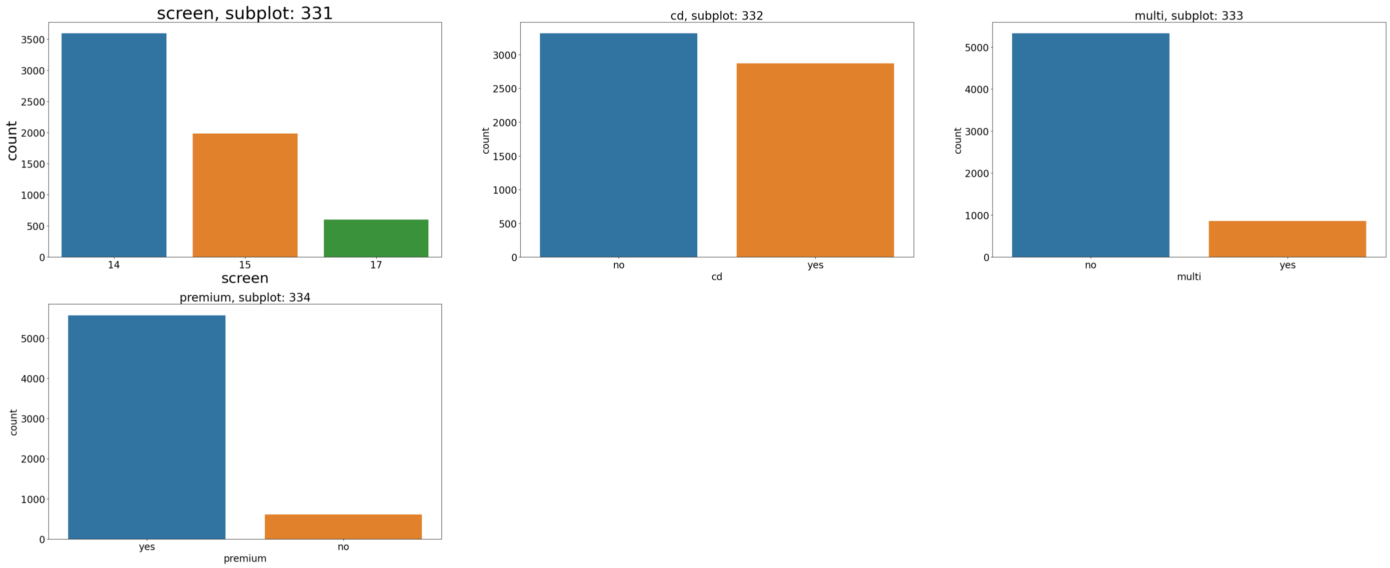
If there is a column which shows too skewed distribution where there is only one dominant bar and the other categories are present in very low numbers. These kind of columns may not be very helpful in machine learning. We confirm this in the correlation analysis section and take a final call to select or reject the column.

Selected Categorical Variables: All the categorical variables are selected for further analysis.

In [14]:
```
cat=["screen","cd","multi","premium"]
```

In [15]:
```
#bivariate plot-political.knowledge
fig = plt.figure(figsize=(50,30))
c = 1
for i in cat:
    plt.subplot(3, 3, c)
    plt.title('{}, subplot: {}{}{}'.format(i, 3, 3, c))
    plt.xlabel(i)
```

```
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    #sns.barplot(el_df[i],el_df['age'])
    sns.countplot(x= ComputerPriceData[i], data=ComputerPriceData)
    plt.rcParams.update({'font.size': 20})
    c = c + 1

plt.show()
```



Outlier treatment

Outliers are extreme values in the data which are far away from most of the values. You can see them as the tails in the histogram.

Outlier must be treated one column at a time. As the treatment will be slightly different for each column.

Why I should treat the outliers?

Outliers bias the training of machine learning models. As the algorithm tries to fit the extreme value, it goes away from majority of the data.

There are below two options to treat outliers in the data.

- Option-1: Delete the outlier Records. Only if there are just few rows lost.
- Option-2: Impute the outlier values with a logical business value

In this data no prominent outliers are present, hence, not treating outlier in this section

Missing values treatment

Missing values are treated for each column separately.

If a column has more than 30% data missing, then missing value treatment cannot be done. That column must be rejected because too much information is missing.

There are below options for treating missing values in data.

- Delete the missing value rows if there are only few records

- Impute the missing values with MEDIAN value for continuous variables
- Impute the missing values with MODE value for categorical variables
- Interpolate the values based on nearby values
- Interpolate the values based on business logic

Checking missing Values

In [16]:
```
# No Misiing Values in the data
ComputerPriceData.isnull().sum()
```

Out[16]:
```
price      0
speed      0
hd         0
ram        0
screen     0
cd         0
multi      0
premium    0
ads        0
trend      0
dtype: int64
```

Feature Selection (Bi-Variate Annalysis)

Now its time to finally choose the best columns(Features) which are correlated to the Target variable. This can be done directly by measuring the correlation values or ANOVA/Chi-Square tests. However, it is always helpful to visualize the relation between the Target variable and each of the predictors to get a better sense of data.

I have listed below the techniques used for visualizing relationship between two variables as well as measuring the strength statistically.

Visual exploration of relationship between variables

- Continuous Vs Continuous ---- Scatter Plot
- Categorical Vs Continuous---- Box Plot
- Categorical Vs Categorical---- Grouped Bar Plots

Statistical measurement of relationship strength between variables

- Continuous Vs Continuous ---- Correlation matrix
- Categorical Vs Continuous---- ANOVA test
- Categorical Vs Categorical--- Chi-Square test

In this case study the Target variable is Continuous, hence below two scenarios will be present

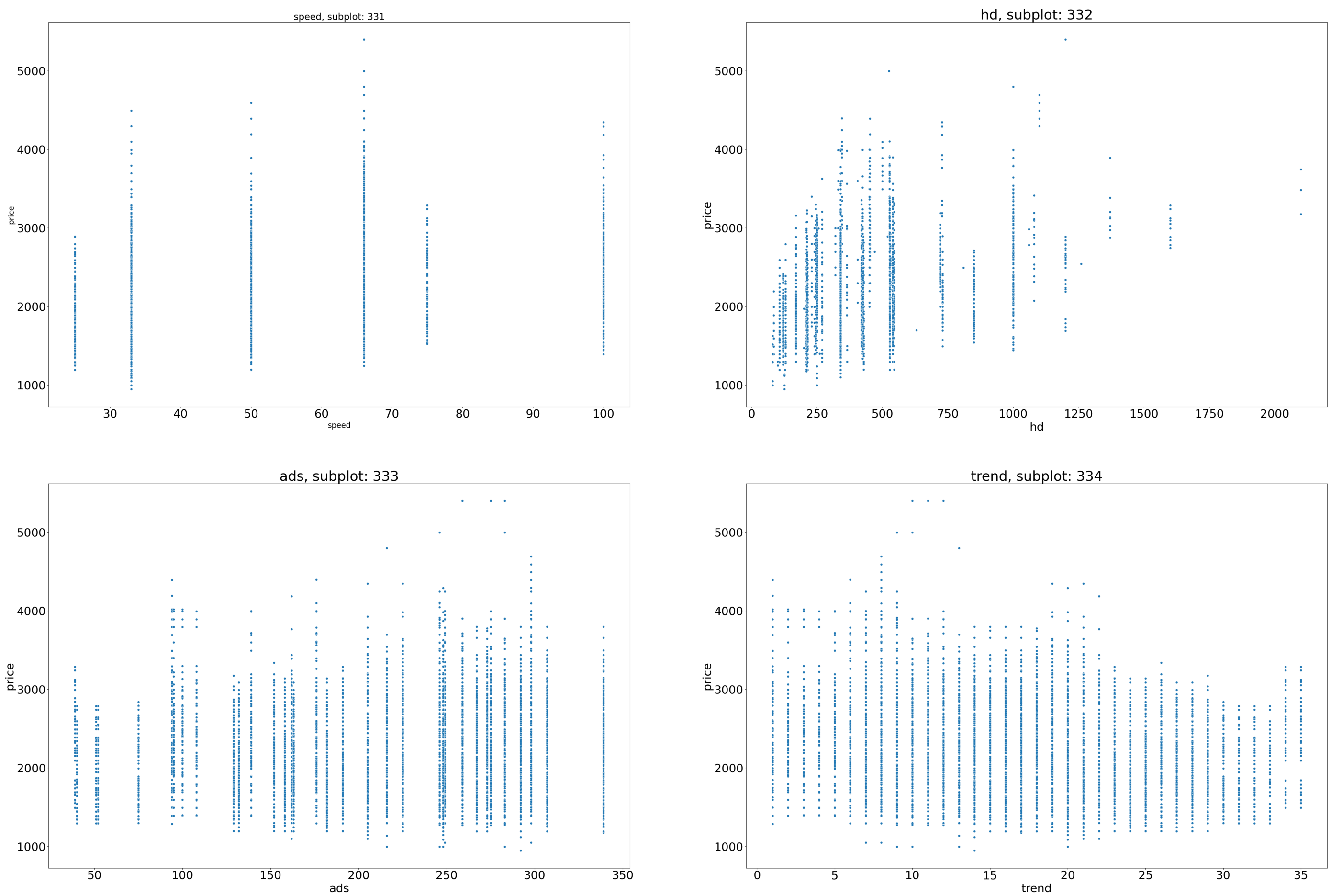
- Continuous Target Variable Vs Continuous Predictor

- Continuous Target Variable Vs Categorical Predictor

Relationship exploration: Continuous Vs Continuous -- Scatter Charts

When the Target variable is continuous and the predictor is also continuous, we can visualize the relationship between the two variables using scatter plot and measure the strength of relation using pearson's correlation value.

In [17]:
```
ContinuousCols=['speed','hd','ads','trend']

fig = plt.figure(figsize=(60,40))
c = 1
for i in ContinuousCols:
    plt.subplot(2, 2, c)
    plt.title('{}, subplot: {}{}{}'.format(i, 3, 3, c))
    plt.xlabel(i)
    plt.xticks(fontsize=30)
    plt.yticks(fontsize=30)
    #sns.barplot(el_df[i],el_df['age'])
    sns.scatterplot(x= ComputerPriceData[i], y="price", data=ComputerPriceData)
    plt.rcParams.update({'font.size': 30})
    c = c + 1

plt.show()
```



Scatter charts interpretation

What should you look for in these scatter charts?

Trend. You should try to see if there is a visible trend or not. There could be three scenarios

Increasing Trend: This means both variables are positively correlated. In simpler terms, they are directly proportional to each other, if one value increases, other also increases. This is good for ML!

Decreasing Trend: This means both variables are negatively correlated. In simpler terms, they are inversely proportional to each other, if one value increases, other decreases. This is also good for ML!

No Trend: You cannot see any clear increasing or decreasing trend. This means there is no correlation between the variables. Hence the predictor cannot be used for ML.

Based on this chart you can get a good idea about the predictor, if it will be useful or not. You confirm this by looking at the correlation value.

Statistical Feature Selection (Continuous Vs Continuous) using Correlation value

Pearson's correlation coefficient can simply be calculated as the covariance between two features x and y (numerator) divided by the product of their standard deviations (denominator):

This value can be calculated only between two numeric columns Correlation between [-1,0) means inversely proportional, the scatter plot will show a downward trend Correlation between (0,1] means directly proportional, the scatter plot will show a upward trend Correlation near {0} means No relationship, the scatter plot will show no clear trend. If Correlation value between two variables is > 0.5 in magnitude, it indicates good relationship the sign does not matter We observe the correlations between Target variable and all other predictor variables(s) to check which columns/features/predictors are actually related to the target variable in question

In [18]:
```
# Calculating correlation matrix
ContinuousCols=['price','speed','hd','ads','trend']

# Creating the correlation matrix
CorrelationData=ComputerPriceData[ContinuousCols].corr()
CorrelationData
```

Out[18]:

	price	hd	ads	trend
price	1.000000	0.428845	0.056434	-0.201662
hd	0.428845	1.000000	-0.323342	0.577599
ads	0.056434	-0.323342	1.000000	-0.320626
trend	-0.201662	0.577599	-0.320626	1.000000

In [19]:
```
# Filtering only those columns where absolute correlation > 0.5 with Target Variable
# reduce the 0.5 threshold if no variable is selected
CorrelationData['price'][abs(CorrelationData['price']) > 0.2 ]
```

Out[19]:
```
price     1.000000
hd        0.428845
trend    -0.201662
Name: price, dtype: float64
```

Final selected Continuous columns:

'speed','hd','trend'

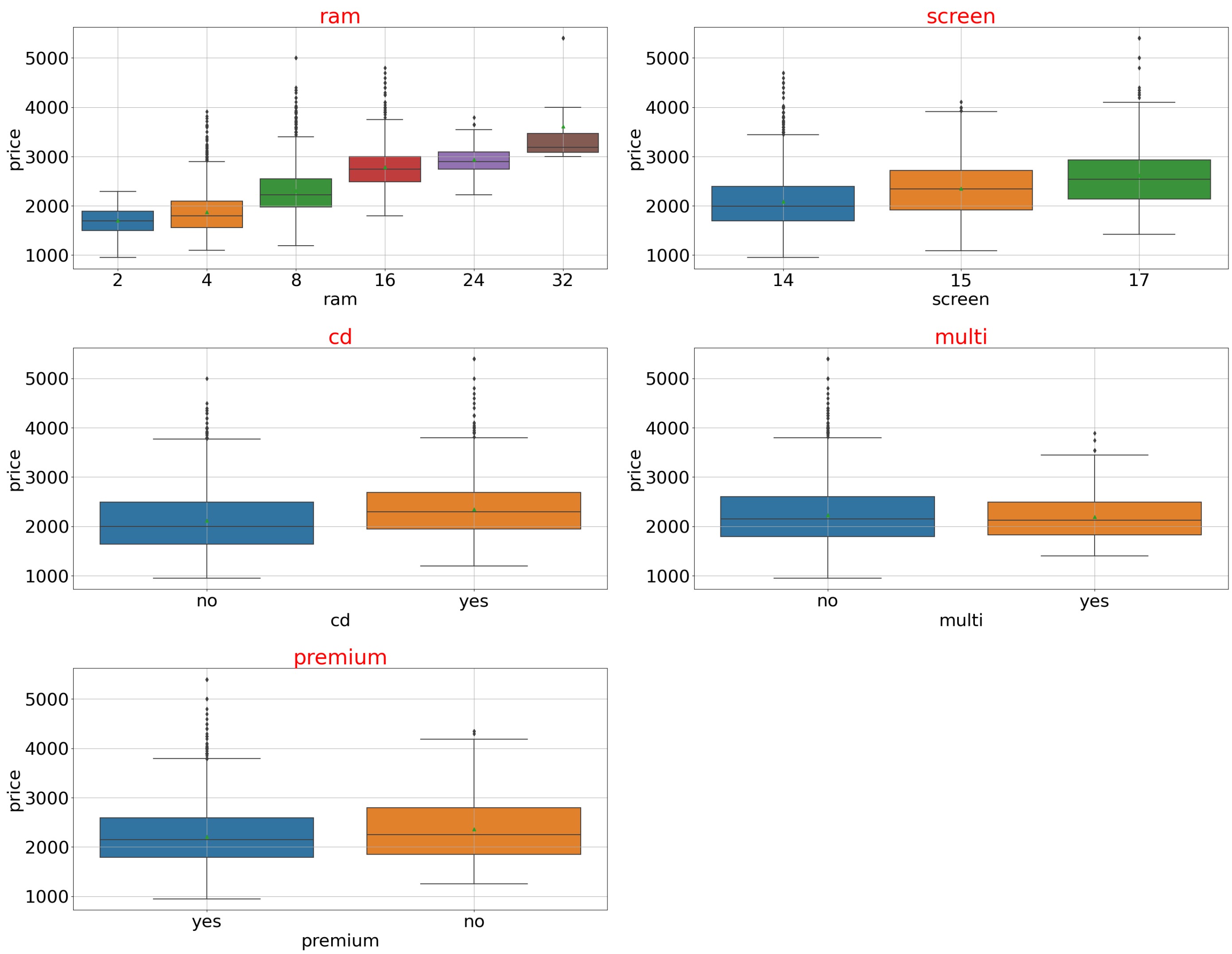
In [ ]:

Relationship exploration: Categorical Vs Continuous -- Box Plots

When the target variable is Continuous and the predictor variable is Categorical we analyze the relation using Boxplots and measure the strength of relation using Anova test

In [20]:

```
ColsToPlot=["ram","screen","cd","multi","premium"]

fig=plt.figure(figsize=(30,30))
for i in range(0,len(ColsToPlot)):
    ax=fig.add_subplot(4,2,i+1)
    sns.boxplot(x=ColsToPlot[i],y='price', data=ComputerPriceData,showmeans=True)
    ax.set_title(ColsToPlot[i],color='Red')
    plt.grid()
plt.tight_layout()
```



Box-Plots interpretation

What should you look for in these box plots?

These plots gives an idea about the data distribution of continuous predictor in the Y-axis for each of the category in the X-Axis.

If the distribution looks similar for each category(Boxes are in the same line), that means the the continuous variable has NO effect on the target variable. Hence, the variables are not correlated to each other.

On the other hand if the distribution is different for each category(the boxes are not in same line!). It hints that these variables might be correlated with price.

In this data, all the categorical predictors looks correlated with the Target variable except "multi", it seems like a border case, as the boxes are close to each other.

We confirm this by looking at the results of ANOVA test below

Statistical Feature Selection (Categorical Vs Continuous) using ANOVA test

Analysis of variance(ANOVA) is performed to check if there is any relationship between the given continuous and categorical variable

- Assumption(H0): There is NO relation between the given variables (i.e. The average(mean) values of the numeric Target variable is same for all the groups in the categorical Predictor variable)
- ANOVA Test result: Probability of H0 being true

In [21]:
```
# Defining a function to find the statistical relationship with all the categorical vari

def FunctionAnova(inpData,TargetVariable,PredictorList):
    from scipy.stats import f_oneway

    # Creating an empty list of final selected predictors
    SelectedPredictors=[]

    print('##### ANOVA Results ##### \n')
    for predictor in PredictorList:
        CategoryGroupLists=inpData.groupby(predictor)[TargetVariable].apply(list)
        AnovaResults = f_oneway(*CategoryGroupLists)

        # If the ANOVA P-Value is <0.05, that means we reject H0
        if (AnovaResults[1] < 0.05):
            print(predictor, 'is correlated with', TargetVariable, '| P-Value:', AnovaRe
            SelectedPredictors.append(predictor)
        else:
            # Accepting the H0 if the P value is more than 0.05
            print(predictor, 'is NOT correlated with', TargetVariable, '| P-Value:', Ano

    return(SelectedPredictors)
```

In [22]:
```
# Calling the function to check which categorical variables are correlated with target
# Calling the function to check which categorical variables are correlated with target

PredictorList=["ram","screen","cd","multi","premium"]
FunctionAnova(inpData=ComputerPriceData,TargetVariable="price",
              PredictorList=PredictorList)
```

```
##### ANOVA Results #####

ram is correlated with price | P-Value: 0.0
screen is correlated with price | P-Value: 1.2830206408407136e-129
cd is correlated with price | P-Value: 8.113565801487017e-55
```

```
multi is NOT correlated with price | P-Value: 0.19076936432204794
premium is correlated with price | P-Value: 2.7969949437607514e-10
```

Out[22]:
```
['ram', 'screen', 'cd', 'premium']
```

Selecting Final Predictors for ML

In [23]:
```
SelectedCols=['speed', 'hd', 'ram', 'screen', 'cd', 'premium',  'trend']
DataForML=ComputerPriceData[SelectedCols]
DataForML.head()
```

Out[23]:

	speed	hd	ram	screen	cd	premium	trend
0	25	80	4	14	no	yes	1
1	33	85	2	14	no	yes	1
2	25	170	4	15	no	yes	1
3	25	170	8	14	no	no	1
4	33	340	16	14	no	yes	1

In [24]:
```
# Saving this final data for reference during deployment
DataForML.to_pickle('DataForML.pkl')
```

Data Pre-processing for Machine Learning

List of steps performed on predictor variables before data can be used for machine learning

1. Converting each Ordinal Categorical columns to numeric
2. Converting Binary nominal Categorical columns to numeric using 1/0 mapping
3. Converting all other nominal categorical columns to numeric using pd.get_dummies()
4. Data Transformation (Optional): Standardization/Normalization/log/sqrt. Important if you are using distance based algorithms like KNN, or Neural Networks

In [25]:
```
DataForML["cd"].replace({"no":0,"yes":1},inplace=True)
DataForML["premium"].replace({"no":0,"yes":1},inplace=True)
```

In [26]:
```
# Adding Target Variable to the data
DataForML['price']=ComputerPriceData['price']

# Printing sample rows
DataForML.head()
```

Out[26]:

	speed	hd	ram	screen	cd	premium	trend	price
0	25	80	4	14	0	1	1	1499
1	33	85	2	14	0	1	1	1795
2	25	170	4	15	0	1	1	1595
3	25	170	8	14	0	0	1	1849
4	33	340	16	14	0	1	1	3295

Machine Learning: Splitting the data into Training and Testing sample

We dont use the full data for creating the model. Some data is randomly selected and kept aside for checking how good the model is. This is known as Testing Data and the remaining data is called Training data on which the model is built. Typically 70% of data is used as Training data and the rest 30% is used as Tesing data.

In [27]:
```
DataForML.columns
```

Out[27]:
```
Index(['speed', 'hd', 'ram', 'screen', 'cd', 'premium', 'trend', 'price'], dtype='objec
t')
```

In [28]:
```
# Separate Target Variable and Predictor Variables
TargetVariable="price"
Predictors=["speed","hd","ram","screen","cd","premium","trend"]

X=DataForML[Predictors].values
y=DataForML[TargetVariable].values

# Split the data into training and testing set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=3)
```

In [29]:
```
# Suppressing Scientific Notation in printing numpy arrays
np.set_printoptions(suppress=True)
X_train[0:7]
```

Out[29]:
```
array([[66, 424, 8, 15, 1, 1, 15],
       [66, 528, 8, 15, 1, 1, 21],
       [66, 528, 8, 14, 1, 1, 23],
       [33, 720, 16, 15, 1, 1, 22],
       [50, 340, 4, 14, 0, 1, 9],
       [66, 528, 4, 14, 0, 1, 30],
       [50, 528, 16, 14, 0, 1, 10]], dtype=object)
```

In [30]:
```
X_train[0:7]
```

Out[30]:
```
array([[66, 424, 8, 15, 1, 1, 15],
       [66, 528, 8, 15, 1, 1, 21],
       [66, 528, 8, 14, 1, 1, 23],
       [33, 720, 16, 15, 1, 1, 22],
       [50, 340, 4, 14, 0, 1, 9],
       [66, 528, 4, 14, 0, 1, 30],
       [50, 528, 16, 14, 0, 1, 10]], dtype=object)
```

Multiple Linear Regression

In [31]:
```
# Multiple Linear Regression
from sklearn.linear_model import LinearRegression
RegModel =LinearRegression()

# Printing all the parameters of Linear regression
print(RegModel)

# Creating the model on Training Data
LREG =RegModel.fit(X_train,y_train)

# Taking the standardized values to original scale
```

```
from sklearn import metrics

# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, LREG.predict(X_train)))

###########################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction

prediction =LREG.predict(X_test)
TestingDataResults=pd.DataFrame(data=X_test , columns =Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[("Predicted"+TargetVariable)]=np.round(prediction)


# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['price']-TestingDataResults['Predictedprice']))/TestingDataResults[

# Printing sample prediction values
print(TestingDataResults[[TargetVariable,'Predicted'+TargetVariable, 'APE']].head())

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlie
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
LinearRegression()
R2 Value: 0.7647567559646572

##### Model Validation and Accuracy Calculations ##########
   price  Predictedprice        APE
0   1994          1922.0   3.610832
1   2990          3292.0  10.100334
2   1899          1738.0   8.478146
3   2989          2626.0  12.144530
4   2795          2630.0   5.903399
Mean Accuracy on test data: 90.28215352461106
Median Accuracy on test data: 92.6299456943367
```

```
Accuracy values for 10-fold Cross Validation:
 [86.25794432 87.50348066 89.56671953 88.36697612 91.01667583 91.95822707
 91.73501234 90.74300352 89.8634562  87.04312012]

Final Average Accuracy of the model: 89.41
```

In [32]:

```
TestingDataResults["AvgPrice"]=round(TestingDataResults["price"].mean(),2)
TestingDataResults.head()
```

Out[32]:

	speed	hd	ram	screen	cd	premium	trend	price	Predictedprice	APE	AvgPrice
0	50	107	2	14	1	1	10	1994	1922.0	3.610832	2225.45
1	66	1000	24	17	1	1	26	2990	3292.0	10.100334	2225.45
2	66	425	8	14	1	1	26	1899	1738.0	8.478146	2225.45
3	66	424	8	14	0	0	17	2989	2626.0	12.144530	2225.45
4	50	528	16	14	0	1	13	2795	2630.0	5.903399	2225.45

In [ ]:

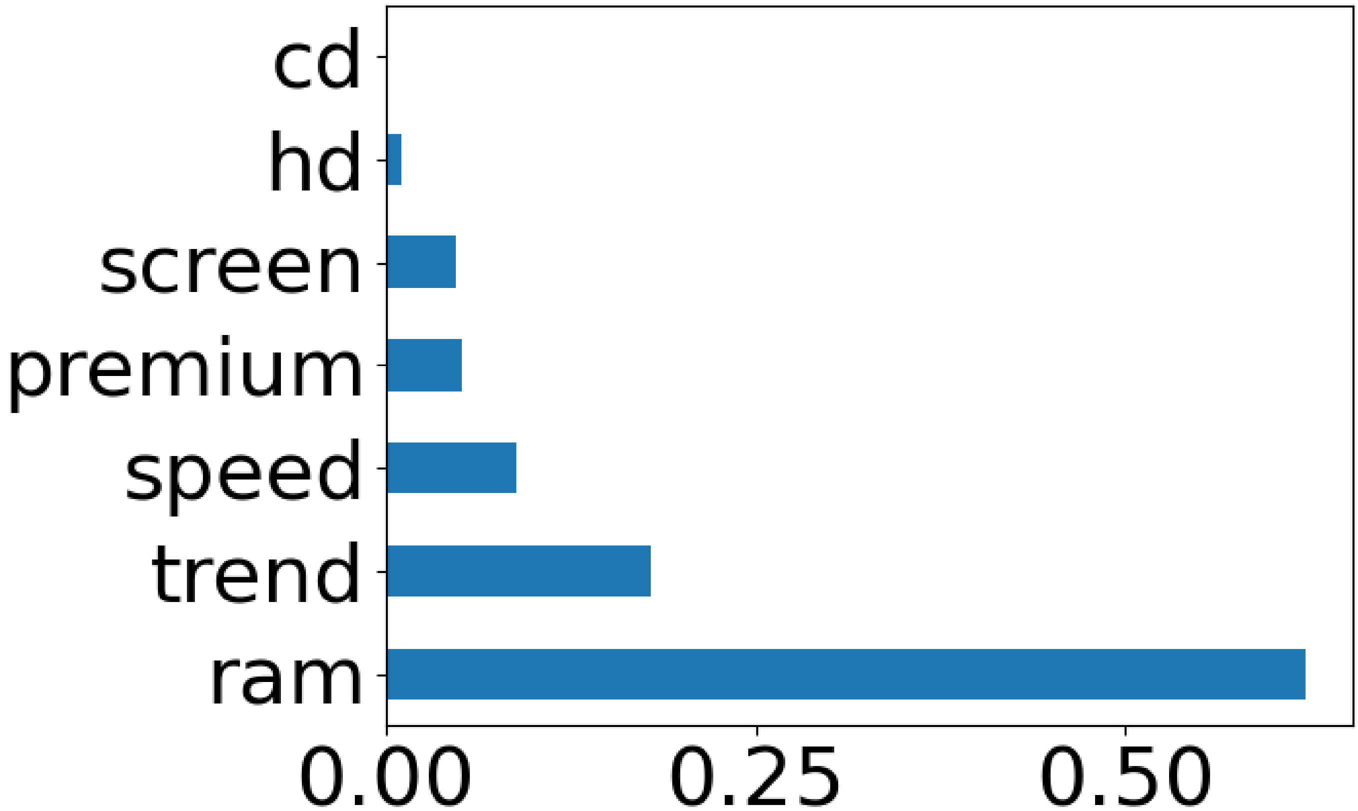
Plotting a Decision Tree

In [33]:

```
# Decision Trees (Multiple if-else statements!)
from sklearn.tree import DecisionTreeRegressor
RegModel = DecisionTreeRegressor(max_depth=4,criterion='poisson')
# Good Range of Max_depth = 2 to 20

# Printing all the parameters of Decision Tree
print(RegModel)

# Creating the model on Training Data
DT=RegModel.fit(X_train,y_train)
prediction=DT.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, DT.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(DT.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

###########################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults[[TargetVariable,'Predicted'+TargetVariable]].head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['price']-TestingDataResults['Predictedprice']))/TestingDataResults[

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])
```

```
Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlie
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

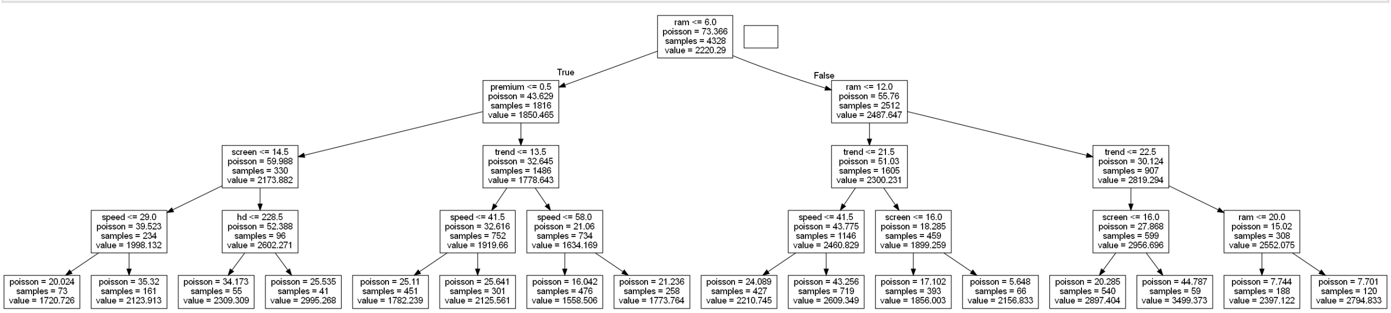
```
DecisionTreeRegressor(criterion='poisson', max_depth=4)
R2 Value: 0.6524331938206238

##### Model Validation and Accuracy Calculations ##########
   price  Predictedprice
0   1994          2126.0
1   2990          2795.0
2   1899          1856.0
3   2989          2609.0
4   2795          2897.0
Mean Accuracy on test data: 88.48804989900583
Median Accuracy on test data: 91.2431941923775

Accuracy values for 10-fold Cross Validation:
 [85.73328704 87.46285233 88.50755154 87.58673482 86.39271565 85.43321056
 83.03607477 86.32402142 88.88745347 73.40905985]

Final Average Accuracy of the model: 85.28
```



Plotting a Decision Tree

In [34]:
```
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data
dot_data = tree.export_graphviz(RegModel, out_file=None,
                                feature_names=Predictors, class_names=True)

# printing the rules
#print(dot_data)

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=9000,height=15000)
# Double click on the graph to zoom in
```

Out[34]:



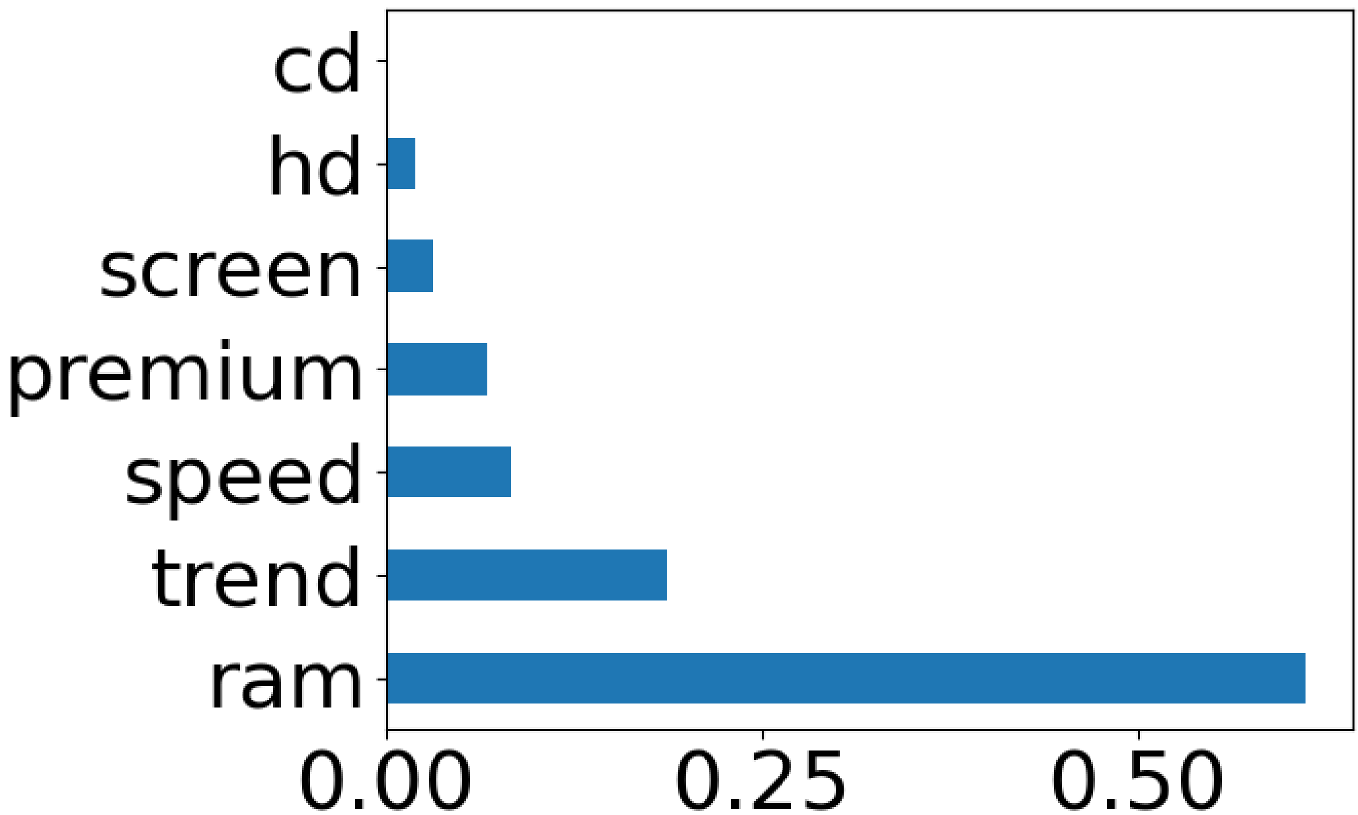
Random Forest

In [35]:
```
# Random Forest (Bagging of multiple Decision Trees)
```

```
from sklearn.ensemble import RandomForestRegressor
RegModel = RandomForestRegressor(max_depth=4, n_estimators=100,criterion='poisson')
# Good range for max_depth: 2-10 and n_estimators: 100-1000

# Printing all the parameters of Random Forest
print(RegModel)

# Creating the model on Training Data
RF=RegModel.fit(X_train,y_train)
prediction=RF.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, RF.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(RF.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

###############################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults[[TargetVariable,'Predicted'+TargetVariable]].head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['price']-TestingDataResults['Predictedprice']))/TestingDataResults[

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlie
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
RandomForestRegressor(criterion='poisson', max_depth=4)
R2 Value: 0.7082311090238718
```

```
##### Model Validation and Accuracy Calculations ##########
   price  Predictedprice
0   1994          2077.0
1   2990          2809.0
2   1899          1888.0
3   2989          2788.0
4   2795          2911.0
Mean Accuracy on test data: 89.42637406772889
Median Accuracy on test data: 92.2361359570662

Accuracy values for 10-fold Cross Validation:
 [86.59311121 88.18100371 89.00877434 88.91750713 87.64575725 86.98314235
 85.07967576 88.60642625 90.53277782 76.88339434]

Final Average Accuracy of the model: 86.84
```

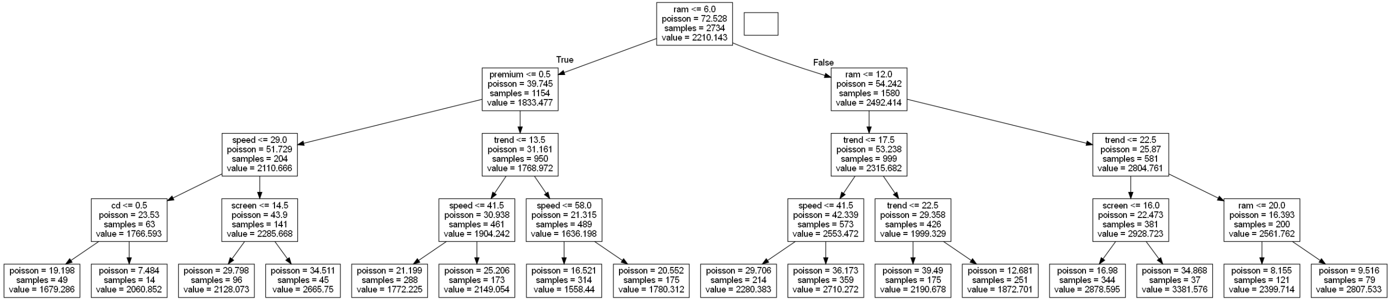


Plotting one of the Decision Trees in Random Forest

In [36]:

```
# Plotting a single Decision Tree from Random Forest
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data for the 6th Decision Tree in Random Forest
dot_data = tree.export_graphviz(RegModel.estimators_[10] , out_file=None, feature_names=

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=9000,height=15000)
# Double click on the graph to zoom in
```

Out[36]:



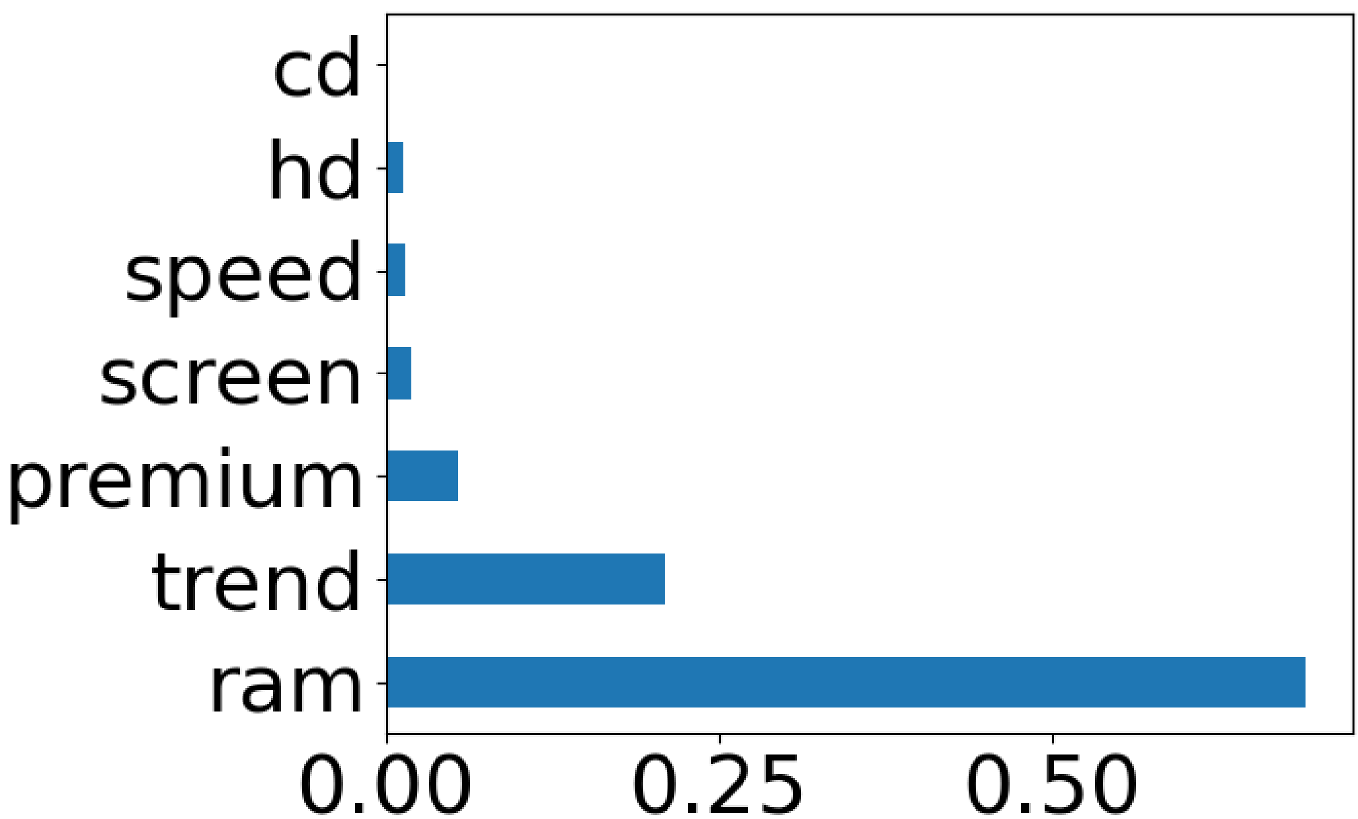
Adaboost

In [37]:

```
# Adaboost (Boosting of multiple Decision Trees)
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

# Choosing Decision Tree with 5 level as the weak learner
# learning_rate between 0.01 to 0.05
# max_depth between 1 to 10
# n_estimators from 100 to 5000
DTR=DecisionTreeRegressor(max_depth=3)
RegModel = AdaBoostRegressor(n_estimators=100, base_estimator=DTR ,learning_rate=0.01)

# Printing all the parameters of Adaboost
print(RegModel)

# Creating the model on Training Data
AB=RegModel.fit(X_train,y_train)
prediction=AB.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, AB.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(AB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

###############################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults[[TargetVariable,'Predicted'+TargetVariable]].head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['price']-TestingDataResults['Predictedprice']))/TestingDataResults[

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlie
print('Median Accuracy on test data:', MedianAccuracy)
```

```
# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3),
                  learning_rate=0.01, n_estimators=100)
R2 Value: 0.5815798561654539

##### Model Validation and Accuracy Calculations ##########
   price  Predictedprice
0   1994          1928.0
1   2990          2583.0
2   1899          1916.0
3   2989          2515.0
4   2795          2994.0
Mean Accuracy on test data: 86.81789302675206
Median Accuracy on test data: 89.92537313432835

Accuracy values for 10-fold Cross Validation:
 [85.58627057 85.82493307 86.91039953 86.24932969 85.79614228 85.60871797
 83.86787383 86.39034238 85.93062521 69.93140518]

Final Average Accuracy of the model: 84.21
```

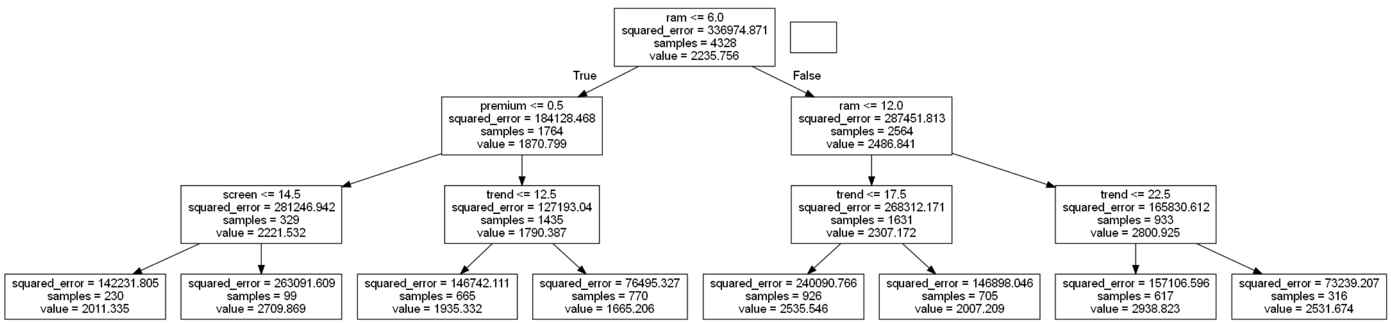


Plotting one of the Decision trees from Adaboost

In [38]:

```
# PLotting 5th single Decision Tree from Adaboost
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data for the 6th Decision Tree in Adaboost
dot_data = tree.export_graphviz(RegModel.estimators_[10] , out_file=None, feature_names=

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=5000,height=5000)
# Double click on the graph to zoom in
```

Out[38]:



XGBoost

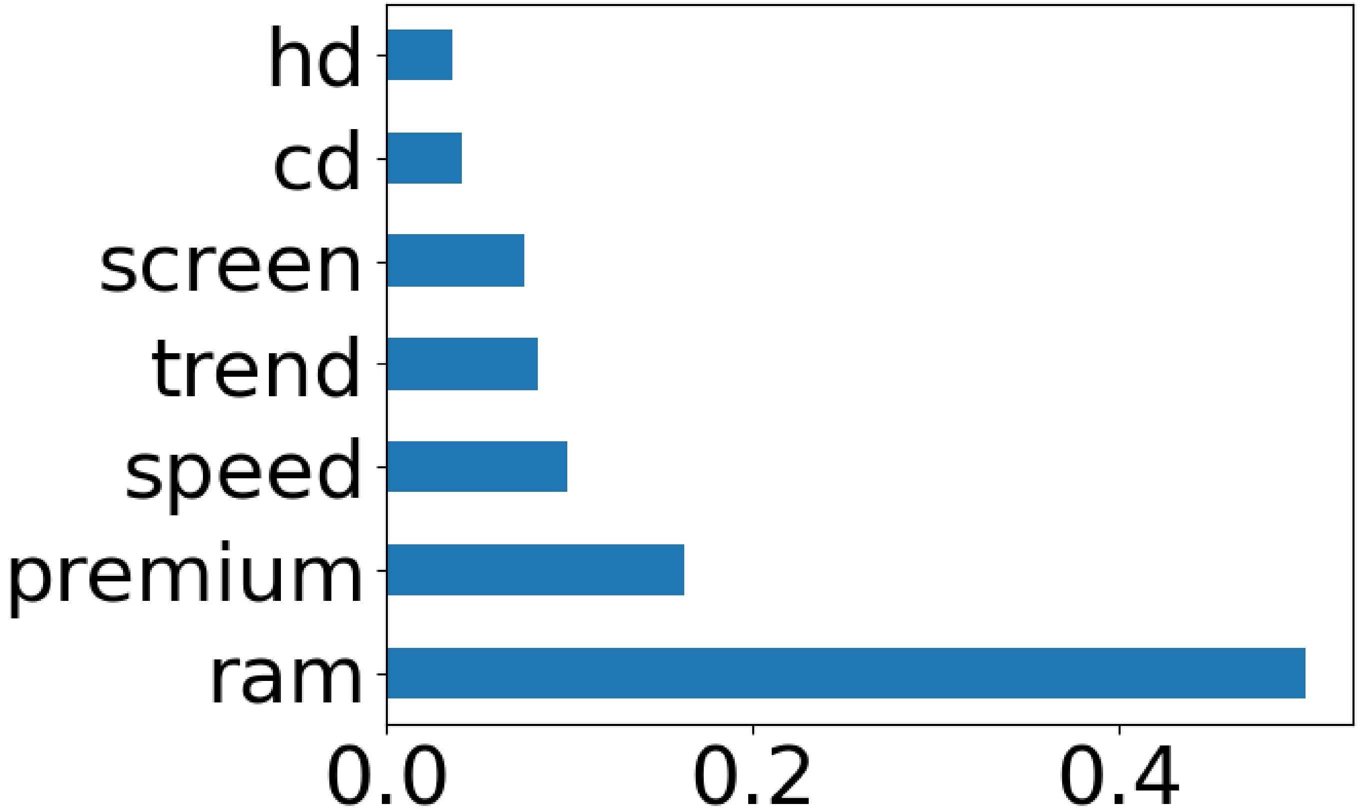
In [39]:

```
from xgboost import XGBRegressor

# Xtreme Gradient Boosting (XGBoost)
from xgboost import XGBRegressor
RegModel=XGBRegressor(max_depth=4,
                      learning_rate=0.3,
                      n_estimators=100,
                      objective='reg:squarederror',
                      booster='gbtree'
                      )

# Printing all the parameters of XGBoost
print(RegModel)

# Creating the model on Training Data
XGB=RegModel.fit(X_train,y_train)
prediction=XGB.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, XGB.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(XGB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')
###############################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
```

```
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults[[TargetVariable,'Predicted'+TargetVariable]].head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['price']-TestingDataResults['Predictedprice']))/TestingDataResults[


MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlie
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.3, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
R2 Value: 0.9444274617654402

##### Model Validation and Accuracy Calculations ##########
   price  Predictedprice
0   1994          2089.0
1   2990          2962.0
2   1899          1750.0
3   2989          3032.0
4   2795          2936.0
Mean Accuracy on test data: 94.71844937038307
Median Accuracy on test data: 95.9610705596107

Accuracy values for 10-fold Cross Validation:
 [88.8796457  93.41750255 94.37502847 92.98844251 92.63057235 93.53147154
 93.19980954 93.69227971 93.7926772  90.74209285]
```

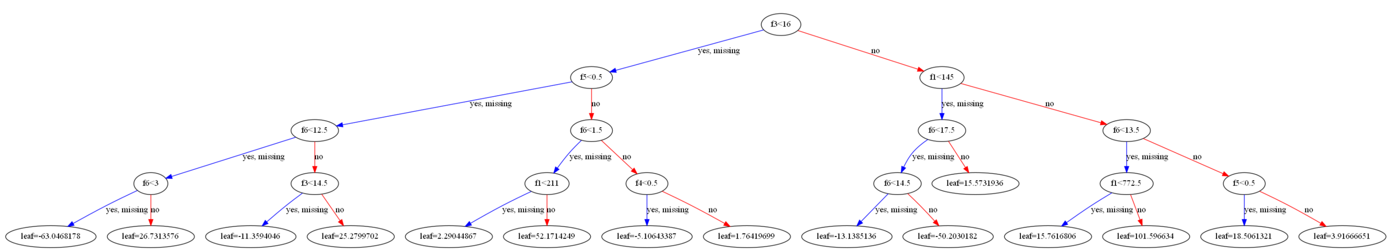
```
Final Average Accuracy of the model: 92.72
```



Plotting a single Decision tree out of XGBoost

In [40]:
```python
from xgboost import plot_tree
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(100, 40))
plot_tree(XGB, num_trees=20, ax=ax)
# Double click on the graph to zoom in
```

Out[40]:
```
<Axes: >
```



In [ ]:
```python

```

Standardization/Normalization of data for KNN Model

However, if you are using KNN or Neural Networks, then this step becomes necessary.

In [41]:
```python
### Sandardization of data ###
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Choose either standardization or Normalization
# On this data Min Max Normalization produced better results

# Choose between standardization and MinMAx normalization
#PredictorScaler=StandardScaler()
```

```
PredictorScaler=MinMaxScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(X)

# Generating the standardized values of X
X=PredictorScalerFit.transform(X)

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=26
```

In [42]:
```
# Sanity check for the sampled data
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(4328, 7)
(4328,)
(1855, 7)
(1855,)
```

KNN

In [43]:
```
# K-Nearest Neighbor(KNN)
from sklearn.neighbors import KNeighborsRegressor
RegModel = KNeighborsRegressor(n_neighbors=3)

# Printing all the parameters of KNN
print(RegModel)

# Creating the model on Training Data
KNN=RegModel.fit(X_train,y_train)
prediction=KNN.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, KNN.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
# The variable importance chart is not available for KNN

###############################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults[[TargetVariable,'Predicted'+TargetVariable]].head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['price']-TestingDataResults['Predictedprice']))/TestingDataResults[

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlie
```

```
print('Median Accuracy on test data:', MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
KNeighborsRegressor(n_neighbors=3)
R2 Value: 0.9360058248200118

##### Model Validation and Accuracy Calculations ##########
   price  Predictedprice
0   1759          1505.0
1   2944          2945.0
2   1395          1460.0
3   1790          1644.0
4   2794          2429.0
Mean Accuracy on test data: 93.02456672225404
Median Accuracy on test data: 94.89499192245557

Accuracy values for 10-fold Cross Validation:
 [90.32359117 91.13354029 92.68634514 91.27258951 92.00574077 93.46149036
 91.72514284 92.20902521 93.41730793 92.48610233]

Final Average Accuracy of the model: 92.07
```

Deployment of the Model

Based on the above trials you select that algorithm which produces the best average accuracy. In this case, multiple algorithms have produced similar kind of average accuracy. Hence, we can choose any one of them.

I am choosing **XGBOOST** as the final model since it is producing the best accuracy on this data.

In order to deploy the model we follow below steps

1. Train the model using 100% data available
2. Save the model as a serialized file which can be stored anywhere
3. Create a python function which gets integrated with front-end(Tableau/Java Website etc.) to take all the inputs and returns the prediction

Choosing only the most important variables

Its beneficial to keep lesser number of predictors for the model while deploying it in production. The lesser predictors you keep, the better because, the model will be less dependent hence, more stable.

This is important specially when the data is high dimensional(too many predictor columns).

In this data, the most important predictor variables are 'trend', 'hd', 'speed', 'ram', and 'screen',premium'.

As these are consistently on top of the variable importance chart for every algorithm. Hence choosing these as final set of predictor variables.

In [105…

```
# Separate Target Variable and Predictor Variables
TargetVariable='price'

# Selecting the final set of predictors for the deployment
# Based on the variable importance charts of multiple algorithms above
Predictors=['trend', 'hd', 'speed', 'ram','screen','premium']

X=DataForML[Predictors].values
y=DataForML[TargetVariable].values

### Sandardization of data ###
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Choose either standardization or Normalization
# On this data Min Max Normalization produced better results

# Choose between standardization and MinMAx normalization
#PredictorScaler=StandardScaler()
PredictorScaler=MinMaxScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(X)

# Generating the standardized values of X
X=PredictorScalerFit.transform(X)

print(X.shape)
print(y.shape)
```

```
(6183, 6)
(6183,)
```

Cross validating the final model accuracy with less predictors

In [106…

```
# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Using final hyperparameters
# Xtreme Gradient Boosting (XGBoost)
#from xgboost import XGBRegressor
RegModel=XGBRegressor(max_depth=4,
                      learning_rate=0.1,
                      n_estimators=150,
                      objective='reg:squarederror',
                      booster='gbtree')
#from sklearn.tree import DecisionTreeRegressor
#RegModel = DecisionTreeRegressor(max_depth=4,criterion='mse')

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
Accuracy values for 10-fold Cross Validation:
 [90.31043417 92.2251721  93.3398937  91.07965386 91.95605523 93.3742595
 92.45807241 92.92745651 93.02217402 89.39000873]

Final Average Accuracy of the model: 92.01
```

In [92]:
```
TestingDataResults.shape
```

Out[92]:
```
(1855, 10)
```

In [107…
```
# Training the model on 100% Data available
Final_XGB_Model=RegModel.fit(X,y)
```

In [108…
```
import pickle
import os

# Saving the Python objects as serialized files can be done using pickle library
# Here let us save the Final model
with open('Final_XGB_Model.pkl', 'wb') as fileWriteStream:
    pickle.dump(Final_XGB_Model, fileWriteStream)
    # Don't forget to close the filestream!
    fileWriteStream.close()

print('pickle file of Predictive Model is saved at Location:',os.getcwd())
```

```
pickle file of Predictive Model is saved at Location: C:\Users\RITWIK\Farooq Sir
```

In [135…
```
# This Function can be called from any from any front end tool/website
def FunctionPredictResult(InputData):
    import pandas as pd
    Num_Inputs=InputData.shape[0]

    # Making sure the input data has same columns as it was used for training the model
    # Also, if standardization/normalization was done, then same must be done for new in

    # Appending the new data with the Training data
    DataForML=pd.read_pickle('DataForML.pkl')
    InputData=InputData.append(DataForML)



    # Maintaining the same order of columns as it was during the model training
    Predictors=['trend', 'hd', 'speed', 'ram','screen','premium']

    # Generating the input values to the model
    X=InputData[Predictors].values[0:Num_Inputs]

    # Generating the standardized values of X since it was done while model training als
    X=PredictorScalerFit.transform(X)

    # Loading the Function from pickle file
    import pickle
    with open('Final_XGB_Model.pkl', 'rb') as fileReadStream:
        PredictionModel=pickle.load(fileReadStream)
        # Don't forget to close the filestream!
        fileReadStream.close()

    # Genprice Predictions
    Prediction=PredictionModel.predict(X)
    PredictionResult=pd.DataFrame(Prediction, columns=['Prediction'])
    return(round(PredictionResult))
```

In [138…
```
# Calling the function for some new data
NewSampleData=pd.DataFrame(
```

```
data=[[1,80,14,4,14,1],
     [1,170,14,8,17,1]],

columns=['trend', 'hd', 'speed','ram','screen','premium'])

print(NewSampleData)
```

```
   trend   hd  speed  ram  screen  premium
0      1   80     14    4      14        1
1      1  170     14    8      17        1
```

In [139…

```
# Calling the Function for prediction
PredictionResult=FunctionPredictResult(InputData= NewSampleData)
PredictionResult
```

Out[139]:

	Prediction
0	1437.0
1	2477.0

Creating the model with few parameters¶

Function for predictions API

In [143…

```
# Creating the function which can take inputs and return predictions
def FunctionGeneratePrediction(inp_trend, inp_hd, inp_speed, inp_ram, inp_screen,inp_pre

    # Creating a data frame for the model input
    SampleInputData=pd.DataFrame(
     data=[[inp_trend, inp_hd, inp_speed, inp_ram, inp_screen,inp_premium]],
     columns=['trend', 'hd', 'speed', 'ram','screen','premium'])

    # Calling the function defined above using the input parameters
    Predictions=FunctionPredictResult(InputData= SampleInputData)

    # Returning the predicted loan status
    return(Predictions.to_json())

# Function call
FunctionGeneratePrediction(inp_trend=1,
                           inp_hd=80,
                           inp_speed=14,
                           inp_ram=4,
                           inp_screen=14,
                           inp_premium=1
                             )
```

Out[143]:

```
'{"Prediction":{"0":1437.0}}'
```

In [ ]: