In [52]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [2]:
```python
df=pd.read_csv("car_price_data2.csv")
```

In [3]:
```python
df.isnull().sum()
```

Out[3]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

In [4]:
```python
df.head()
```

Out[4]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

In [5]:
```
df['car_age']=2021-df['Year']
```

In [6]:
```
df.drop(labels='Year',axis=1,inplace=True)
```

In [7]:
```
df.head()
```

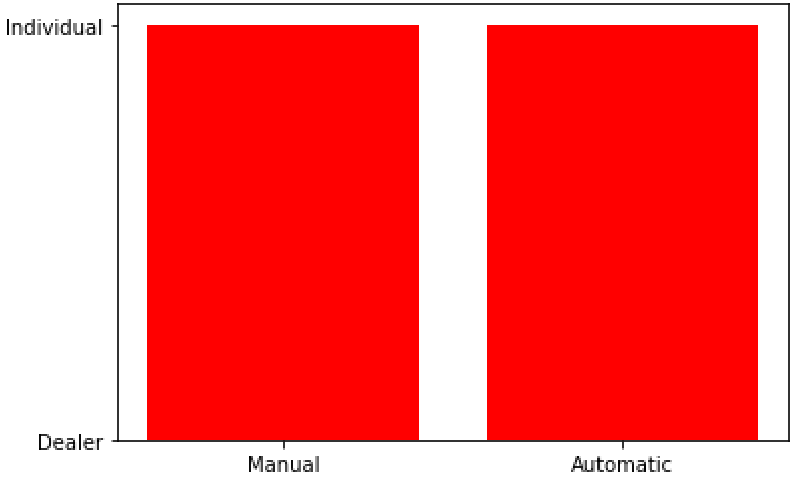
Out[7]:

	Car_Name	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	car_age
0	ritz	3.35	5.59	27000	Petrol	Dealer	Manual	0	7
1	sx4	4.75	9.54	43000	Diesel	Dealer	Manual	0	8
2	ciaz	7.25	9.85	6900	Petrol	Dealer	Manual	0	4
3	wagon r	2.85	4.15	5200	Petrol	Dealer	Manual	0	10
4	swift	4.60	6.87	42450	Diesel	Dealer	Manual	0	7

SOME GRAPHS

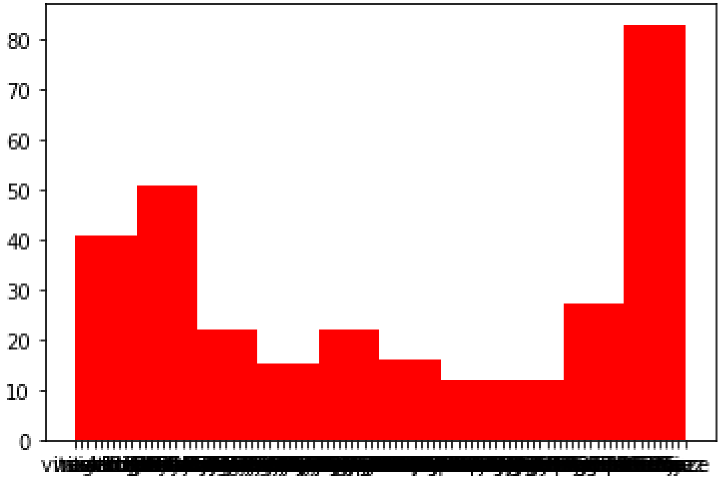
In [8]:
```
plt.bar(df['Transmission'],df['Seller_Type'],color='r') #BARGRAPH
```

Out[8]: <BarContainer object of 301 artists>



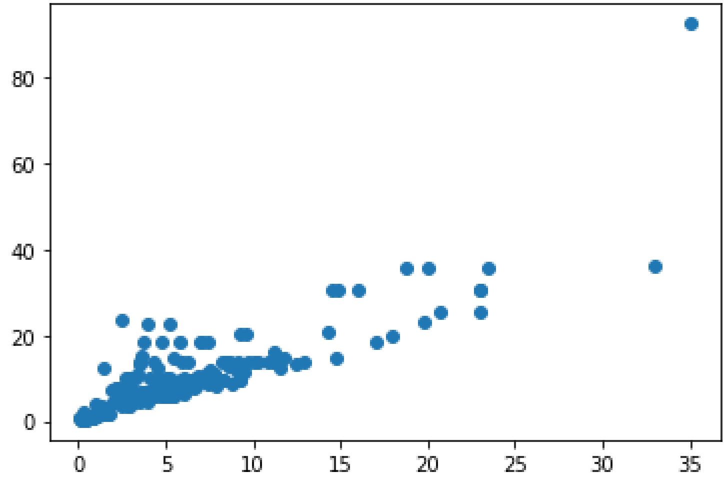
In [9]:
```
plt.hist(df['Car_Name'],color='r')  #HISTOGRAM
```

Out[9]:
```
(array([41., 51., 22., 15., 22., 16., 12., 12., 27., 83.]),
 array([ 0. ,  9.7, 19.4, 29.1, 38.8, 48.5, 58.2, 67.9, 77.6, 87.3, 97. ]),
 <BarContainer object of 10 artists>)
```



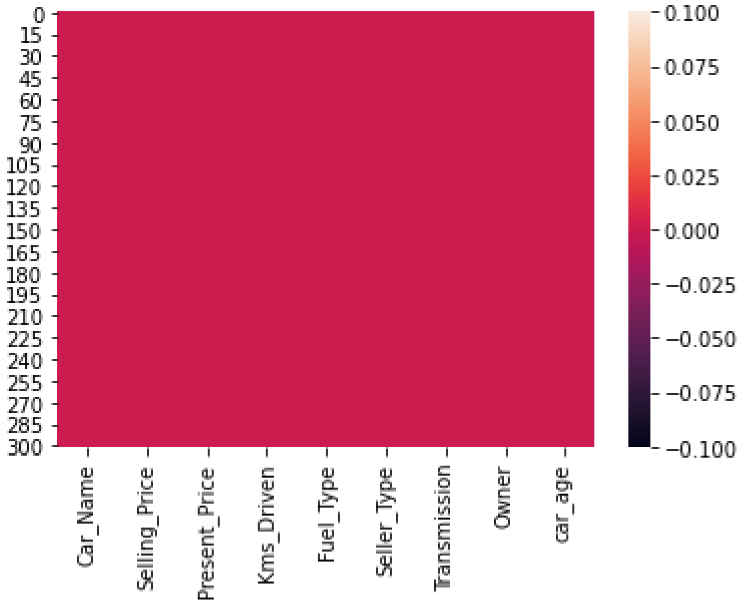
In [10]:
```
plt.scatter(df['Selling_Price'],df['Present_Price']) #SCATTERPLOT
```

Out[10]: <matplotlib.collections.PathCollection at 0x24e516cfa00>



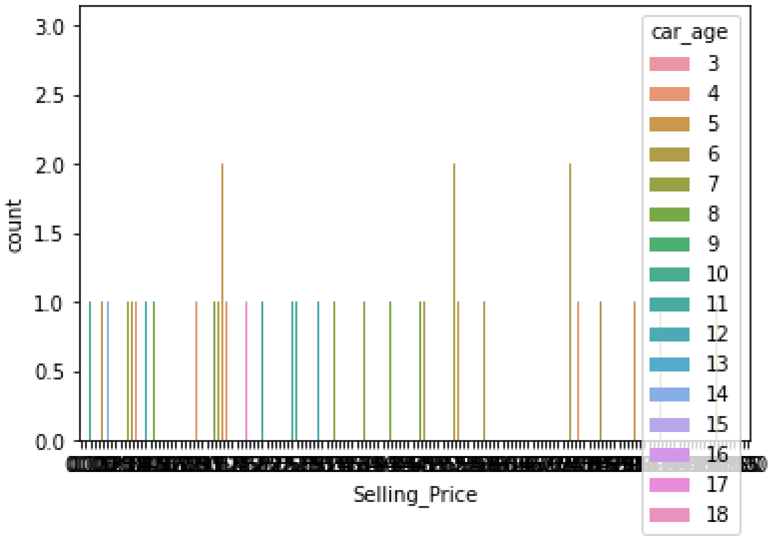
In [11]:
```
sns.heatmap(df.isnull())  #SEABORN
```

Out[11]: <AxesSubplot:>



In [12]:
```
sns.countplot(x='Selling_Price',hue='car_age',data=df)
```

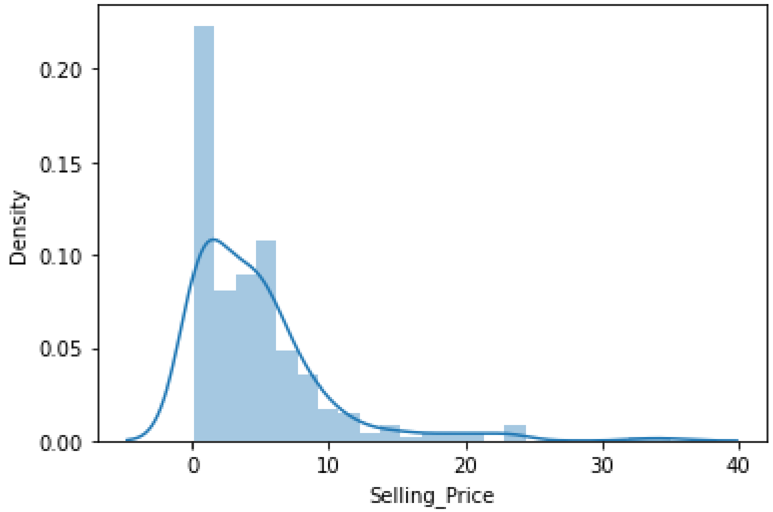
Out[12]: <AxesSubplot:xlabel='Selling_Price', ylabel='count'>



In [13]:
```
left=df[df["Selling_Price"]==1]                          #DISPLOT
notleft=df[df["Selling_Price"]==0]
sns.distplot(df["Selling_Price"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[13]: <AxesSubplot:xlabel='Selling_Price', ylabel='Density'>



DEALING WITH CATEGORICAL VARIABLES

In [14]:
```
df['Fuel_Type'].unique()
```
Out[14]:
```
array(['Petrol', 'Diesel', 'CNG'], dtype=object)
```
In [15]:
```
df['Seller_Type'].unique()
```
Out[15]:
```
array(['Dealer', 'Individual'], dtype=object)
```
In [16]:
```
df['Transmission'].unique()
```
Out[16]:
```
array(['Manual', 'Automatic'], dtype=object)
```

In [17]:
```python
df['Car_Name'].unique()
```

Out[17]:
```
array(['ritz', 'sx4', 'ciaz', 'wagon r', 'swift', 'vitara brezza',
       's cross', 'alto 800', 'ertiga', 'dzire', 'alto k10', 'ignis',
       '800', 'baleno', 'omni', 'fortuner', 'innova', 'corolla altis',
       'etios cross', 'etios g', 'etios liva', 'corolla', 'etios gd',
       'camry', 'land cruiser', 'Royal Enfield Thunder 500',
       'UM Renegade Mojave', 'KTM RC200', 'Bajaj Dominar 400',
       'Royal Enfield Classic 350', 'KTM RC390', 'Hyosung GT250R',
       'Royal Enfield Thunder 350', 'KTM 390 Duke ',
       'Mahindra Mojo XT300', 'Bajaj Pulsar RS200',
       'Royal Enfield Bullet 350', 'Royal Enfield Classic 500',
       'Bajaj Avenger 220', 'Bajaj Avenger 150', 'Honda CB Hornet 160R',
       'Yamaha FZ S V 2.0', 'Yamaha FZ 16', 'TVS Apache RTR 160',
       'Bajaj Pulsar 150', 'Honda CBR 150', 'Hero Extreme',
       'Bajaj Avenger 220 dtsi', 'Bajaj Avenger 150 street',
       'Yamaha FZ  v 2.0', 'Bajaj Pulsar  NS 200', 'Bajaj Pulsar 220 F',
       'TVS Apache RTR 180', 'Hero Passion X pro', 'Bajaj Pulsar NS 200',
       'Yamaha Fazer ', 'Honda Activa 4G', 'TVS Sport ',
       'Honda Dream Yuga ', 'Bajaj Avenger Street 220',
       'Hero Splender iSmart', 'Activa 3g', 'Hero Passion Pro',
       'Honda CB Trigger', 'Yamaha FZ S ', 'Bajaj Pulsar 135 LS',
       'Activa 4g', 'Honda CB Unicorn', 'Hero Honda CBZ extreme',
       'Honda Karizma', 'Honda Activa 125', 'TVS Jupyter',
       'Hero Honda Passion Pro', 'Hero Splender Plus', 'Honda CB Shine',
       'Bajaj Discover 100', 'Suzuki Access 125', 'TVS Wego',
       'Honda CB twister', 'Hero Glamour', 'Hero Super Splendor',
       'Bajaj Discover 125', 'Hero Hunk', 'Hero  Ignitor Disc',
       'Hero  CBZ Xtreme', 'Bajaj  ct 100', 'i20', 'grand i10', 'i10',
       'eon', 'xcent', 'elantra', 'creta', 'verna', 'city', 'brio',
       'amaze', 'jazz'], dtype=object)
```

In [18]:
```python
df.drop(labels='Car_Name',axis=1,inplace=True)
```

In [19]:
```python
df.head()
```

Out[19]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	car_age
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	7
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	8
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	4
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	10
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	7

In [20]:
```python
clean_data=pd.get_dummies(df,drop_first=True)
clean_data.head()
```

Out[20]:

	Selling_Price	Present_Price	Kms_Driven	Owner	car_age	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Manual
0	3.35	5.59	27000	0	7	0	1	0	1
1	4.75	9.54	43000	0	8	1	0	0	1
2	7.25	9.85	6900	0	4	0	1	0	1
3	2.85	4.15	5200	0	10	0	1	0	1
4	4.60	6.87	42450	0	7	1	0	0	1

CKECKING MULTI COLINEARITY

In [21]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [22]:
```
variables=clean_data[['Present_Price','Kms_Driven','Owner','car_age','Fuel_Type_Diesel','Fuel_Type_Petrol','Seller_Type_I
vif=pd.DataFrame()
vif['VIF']=[variance_inflation_factor(variables.values,i) for i in range(variables.shape[1])]
vif['Features']=variables.columns
vif
```

Out[22]:

	VIF	Features
0	3.204463	Present_Price
1	2.892740	Kms_Driven
2	1.087681	Owner
3	10.831000	car_age
4	4.891105	Fuel_Type_Diesel
5	14.342446	Fuel_Type_Petrol
6	2.230725	Seller_Type_Individual
7	8.392371	Transmission_Manual

In [23]:
```
data_no_multicolinearity=clean_data.drop('Fuel_Type_Petrol',axis=1)
```

In [24]:
```
variables=clean_data[['Present_Price','Kms_Driven','Owner','car_age','Fuel_Type_Diesel','Seller_Type_Individual','Transm
vif=pd.DataFrame()
vif['VIF']=[variance_inflation_factor(variables.values,i) for i in range(variables.shape[1])]
vif['Features']=variables.columns
vif
```

Out[24]:

	VIF	Features
0	2.544336	Present_Price
1	2.886452	Kms_Driven
2	1.082447	Owner
3	8.713539	car_age
4	1.706132	Fuel_Type_Diesel
5	1.904835	Seller_Type_Individual
6	4.666095	Transmission_Manual

In [25]:
```
data_no_multicolinearity=clean_data.drop('car_age',axis=1)
```

In [26]:
```
variables=clean_data[['Present_Price','Kms_Driven','Owner','Fuel_Type_Diesel','Seller_Type_Individual','Transmission_Manu
vif=pd.DataFrame()
vif['VIF']=[variance_inflation_factor(variables.values,i) for i in range(variables.shape[1])]
vif['Features']=variables.columns
vif
```

Out[26]:

	VIF	Features
0	2.200428	Present_Price
1	1.883557	Kms_Driven
2	1.065887	Owner
3	1.669188	Fuel_Type_Diesel
4	1.748669	Seller_Type_Individual
5	2.465705	Transmission_Manual

In [27]:
```
data_no_multicolinearity.head()
```

Out[27]:

	Selling_Price	Present_Price	Kms_Driven	Owner	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Manual
0	3.35	5.59	27000	0	0	1	0	1
1	4.75	9.54	43000	0	1	0	0	1
2	7.25	9.85	6900	0	0	1	0	1
3	2.85	4.15	5200	0	0	1	0	1
4	4.60	6.87	42450	0	1	0	0	1

In [28]:
```
dwm=data_no_multicolinearity.drop('Fuel_Type_Petrol',axis=1)
dwm.head()
```

Out[28]:

	Selling_Price	Present_Price	Kms_Driven	Owner	Fuel_Type_Diesel	Seller_Type_Individual	Transmission_Manual
0	3.35	5.59	27000	0	0	0	1
1	4.75	9.54	43000	0	1	0	1
2	7.25	9.85	6900	0	0	0	1
3	2.85	4.15	5200	0	0	0	1
4	4.60	6.87	42450	0	1	0	1

In [29]:
```
x=data_no_multicolinearity.drop('Selling_Price',axis=1)
```

In [30]:
```
y=data_no_multicolinearity['Selling_Price']
```

FEATURE SCALING

In [31]:
```
from sklearn.preprocessing import StandardScaler
scalar=StandardScaler()
scalar.fit(x[['Present_Price','Kms_Driven']])
```

Out[31]:
```
StandardScaler()
```

In [32]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

LINEAR REG

In [33]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
```

In [34]:
```
y_pred
```

Out[34]:
```
array([ 6.88964206,  0.96164601,  3.88984693,  8.36915728, 14.02105143,
        4.39441057,  4.13182723,  0.94941662,  6.05140004,  5.19658685,
        3.47598139,  1.1263384 ,  3.98364301,  7.71311028,  7.84649064,
       13.29880406,  7.11197167,  4.01657233,  0.56173689,  1.47029759,
        6.18974754,  2.776493  ,  7.05249176,  7.22235454, -0.1797864 ,
        0.98802991, -0.47264098,  0.75781258,  0.8861119 ,  8.70752758,
        4.32302694,  7.37294654,  0.63949642,  7.52199065,  4.7588688 ,
        1.18695626,  4.90138059,  6.68329541, -1.02977555,  8.77280008,
        8.28768365, 20.23172023,  4.30469526,  2.70694603,  6.66802818,
        9.16239515,  0.36592768,  1.10496168,  5.03205302,  7.13496559,
        8.58800249,  3.61101905,  4.70045817, 20.10841249,  0.99582927,
        0.81892049,  0.73988511,  2.69648666,  3.33467925,  0.29032312,
        6.18858764])
```

In [35]:
```
y_test
```

Out[35]:
```
223     8.25
150     0.50
226     5.25
296     9.50
52     18.00
       ...  
137     0.65
227     2.55
26      4.15
106     1.35
92      3.51
Name: Selling_Price, Length: 61, dtype: float64
```

In [55]:
```
y_pred2=df.predict(x_test)
y_pred2
```

Out[55]:
```
array([ 6.88964206,  0.96164601,  3.88984693,  8.36915728, 14.02105143,
        4.39441057,  4.13182723,  0.94941662,  6.05140004,  5.19658685,
        3.47598139,  1.1263384 ,  3.98364301,  7.71311028,  7.84649064,
       13.29880406,  7.11197167,  4.01657233,  0.56173689,  1.47029759,
        6.18974754,  2.776493  ,  7.05249176,  7.22235454, -0.1797864 ,
        0.98802991, -0.47264098,  0.75781258,  0.8861119 ,  8.70752758,
        4.32302694,  7.37294654,  0.63949642,  7.52199065,  4.7588688 ,
        1.18695626,  4.90138059,  6.68329541, -1.02977555,  8.77280008,
        8.28768365, 20.23172023,  4.30469526,  2.70694603,  6.66802818,
        9.16239515,  0.36592768,  1.10496168,  5.03205302,  7.13496559,
        8.58800249,  3.61101905,  4.70045817, 20.10841249,  0.99582927,
        0.81892049,  0.73988511,  2.69648666,  3.33467925,  0.29032312,
        6.18858764])
```

In [36]:
```
from sklearn.metrics import mean_squared_error,r2_score
r_squared=r2_score(y_test,y_pred)
r_squared
```

Out[36]:
```
0.8795077661641193
```

In [37]:
```
mse=mean_squared_error(y_test,y_pred)
```

In [38]:
```
mse
```

Out[38]:
```
3.0457205799138443
```

DECISION TREE

In [39]:
```
from sklearn.tree import DecisionTreeRegressor,plot_tree
```

In [40]:
```
dr=DecisionTreeRegressor(random_state=0)
dr.fit(x_train,y_train)
```

Out[40]:
```
DecisionTreeRegressor(random_state=0)
```

In [41]:
```
y_pred1=dr.predict(x_test)
y_pred1
```

Out[41]:
```
array([ 4.95 ,  0.4  ,  4.4  ,  7.25 , 14.25 ,  5.3  ,  2.9  ,  0.25 ,
        5.15 ,  5.225,  2.   ,  0.9  ,  4.85 ,  6.7  ,  7.75 , 14.25 ,
        6.4  ,  3.45 ,  0.45 ,  1.65 ,  2.1  ,  4.9  ,  5.225,  9.7  ,
        0.2  ,  0.4  ,  0.2  ,  0.45 ,  0.45 ,  3.8  ,  3.9  ,  5.95 ,
        0.45 ,  6.5  ,  4.1  ,  1.05 ,  6.25 ,  2.65 ,  0.2  , 11.25 ,
        7.25 , 23.   ,  4.9  ,  4.4  ,  5.5  ,  8.4  ,  0.5  ,  0.4  ,
        5.   ,  7.75 ,  8.99 ,  3.1  ,  5.   , 23.   ,  1.25 ,  1.1  ,
        0.55 ,  2.9  ,  4.   ,  3.   ,  5.5  ])
```

In [42]:
```
y_test
```

Out[42]:
```
223     8.25
150     0.50
226     5.25
296     9.50
52     18.00
       ...
137     0.65
227     2.55
26      4.15
106     1.35
92      3.51
Name: Selling_Price, Length: 61, dtype: float64
```

In [43]:
```
r_squared=r2_score(y_test,y_pred1)
r_squared
```

Out[43]:
```
0.9079354451502754
```

In [44]:
```
mse=mean_squared_error(y_test,y_pred1)
mse
```

Out[44]:
```
2.3271450819672133
```

In [45]:
```
df=dr.fit(x_test,y_pred)
plt.figure(figsize=(15,7))
plot_tree(df,filled=True)
```

Out[45]:
```
[Text(486.0531496062992, 365.88461538461536, 'X[0] <= 7.775\nmse = 18.779\nsamples = 61\nvalue = 4.944'),
 Text(265.2696850393701, 336.6138461538461, 'X[0] <= 3.94\nmse = 3.157\nsamples = 34\nvalue = 2.081'),
 Text(187.8307086614173, 307.3430769230769, 'X[1] <= 25500.0\nmse = 0.363\nsamples = 19\nvalue = 0.64'),
 Text(125.22047244094487, 278.0723076923077, 'X[1] <= 10850.0\nmse = 0.077\nsamples = 13\nvalue = 0.931'),
 Text(79.08661417322834, 248.80153846153846, 'X[0] <= 1.66\nmse = 0.028\nsamples = 7\nvalue = 1.113'),
 Text(65.90551181102362, 219.5307692307692, 'X[0] <= 0.895\nmse = 0.008\nsamples = 6\nvalue = 1.053'),
 Text(39.54330708661417, 190.26, 'X[0] <= 0.833\nmse = 0.0\nsamples = 3\nvalue = 0.966'),
 Text(26.362204724409448, 160.98923076923077, 'X[0] <= 0.688\nmse = 0.0\nsamples = 2\nvalue = 0.956'),
 Text(13.181102362204724, 131.71846153846153, 'mse = 0.0\nsamples = 1\nvalue = 0.949'),
 Text(39.54330708661417, 131.71846153846153, 'mse = 0.0\nsamples = 1\nvalue = 0.962'),
 Text(52.724409448818896, 160.98923076923077, 'mse = -0.0\nsamples = 1\nvalue = 0.988'),
 Text(92.26771653543307, 190.26, 'X[1] <= 7350.0\nmse = 0.001\nsamples = 3\nvalue = 1.139'),
 Text(79.08661417322834, 160.98923076923077, 'X[0] <= 1.06\nmse = 0.0\nsamples = 2\nvalue = 1.116'),
 Text(65.90551181102362, 131.71846153846153, 'mse = 0.0\nsamples = 1\nvalue = 1.126'),
 Text(92.26771653543307, 131.71846153846153, 'mse = 0.0\nsamples = 1\nvalue = 1.105'),
 Text(105.44881889763779, 160.98923076923077, 'mse = 0.0\nsamples = 1\nvalue = 1.187'),
 Text(92.26771653543307, 219.5307692307692, 'mse = 0.0\nsamples = 1\nvalue = 1.47'),
 Text(171.35433070866142, 248.80153846153846, 'X[0] <= 2.475\nmse = 0.049\nsamples = 6\nvalue = 0.718'),
 Text(158.17322834645668, 219.5307692307692, 'X[0] <= 1.35\nmse = 0.015\nsamples = 5\nvalue = 0.804'),
```

RANDOM FOREST

In [46]:
```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
rf.fit(x_train,y_train)
y_pred=rf.predict(x_test)
```

In [47]:
```
y_pred
```

Out[47]:
```
array([ 5.057     ,  0.5375    ,  4.5135    ,  8.08695   , 13.402     ,
        4.939     ,  3.567     ,  0.4035    ,  4.4535    ,  5.37258333,
        2.619     ,  0.8335    ,  4.6505    ,  8.104     ,  7.4685    ,
       13.4874    ,  7.18      ,  3.7565    ,  0.4919    ,  1.624     ,
        3.484     ,  4.8705    ,  5.89860833,  9.7695    ,  0.2121    ,
        0.5903    ,  0.2767    ,  0.6487    ,  0.522     ,  5.656     ,
        3.4545    ,  5.7485    ,  0.4886    ,  7.0495    ,  4.356     ,
        1.0995    ,  5.9515    ,  4.616075  ,  0.2868    ,  8.7489    ,
        7.8433    , 23.6875    ,  4.9075    ,  4.27775   ,  5.8575    ,
       10.98      ,  0.3249    ,  0.6545    ,  4.86233333,  6.41249167,
        9.2846    ,  3.068     ,  4.87116667, 23.7875    ,  1.1437    ,
        1.1101    ,  0.56331667,  2.8065    ,  3.37      ,  2.5781    ,
        5.539     ])
```

In [48]:
```
r_squared=r2_score(y_test,y_pred)
r_squared
```

Out[48]:
```
0.9210630259934256
```

In [49]:
```
mse=mean_squared_error(y_test,y_pred)
mse
```

Out[49]:
```
1.995315038937841
```