In [89]:
```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC, LinearSVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, roc_curve, auc, classificatio
from sklearn import svm
# Visualisation libraries

## Text
from colorama import Fore, Back, Style
from IPython.display import Image, display, Markdown, Latex, clear_output

## plotly
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
import plotly.offline as py
from plotly.subplots import make_subplots
import plotly.express as px

## seaborn
import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("paper", rc={"font.size":12,"axes.titlesize":14,"axes.label

## matplotlib
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties
from matplotlib.patches import Ellipse, Polygon
import matplotlib.gridspec as gridspec
import matplotlib.colors
from pylab import rcParams
from matplotlib.font_manager import FontProperties
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
plt.style.use('seaborn-whitegrid')
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (17, 6)
mpl.rcParams['axes.labelsize'] = 14
mpl.rcParams['xtick.labelsize'] = 12
mpl.rcParams['ytick.labelsize'] = 12
mpl.rcParams['text.color'] = 'k'
%matplotlib inline

import sklearn.linear_model as skl_lm
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, LeaveOneOut, KFold, c
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, exp
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegress
from sklearn.metrics import confusion_matrix, mean_squared_error
from sklearn import tree
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv('desktop/Carseats.csv')
```

In [3]:
```
df
```

Out[3]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban
0	9.50	138	73	11	276	120	Bad	42	17	Yes
1	11.22	111	48	16	260	83	Good	65	10	Yes
2	10.06	113	35	10	269	80	Medium	59	12	Yes
3	7.40	117	100	4	466	97	Medium	55	14	Yes
4	4.15	141	64	3	340	128	Bad	38	13	Yes
...	...	...	...	...	...	...	...	...	...	...
395	12.57	138	108	17	203	128	Good	33	14	Yes
396	6.14	139	23	3	37	120	Medium	55	11	No
397	7.41	162	26	12	368	159	Medium	40	18	Yes
398	5.94	100	79	7	284	95	Bad	50	12	Yes
399	9.71	134	37	0	27	120	Good	49	16	Yes

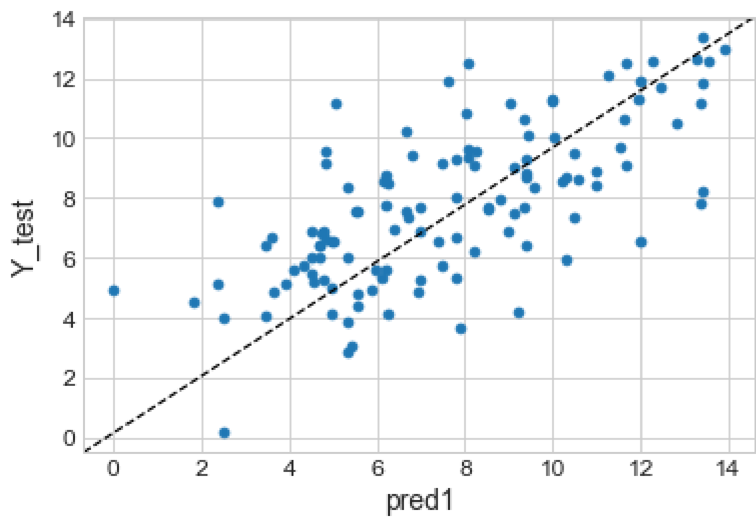
400 rows × 11 columns

In [4]:
```
df['Urban'] = df['Urban'].map({'Yes':0, 'No':1})
df['US'] = df['US'].map({'Yes':0, 'No':1})
df['ShelveLoc'] = df['ShelveLoc'].map({'Good':0, 'Medium':1,'Bad':2})
```

In [5]:
```
X = df.drop('Sales', axis=1)
Y = df.Sales
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7, r


regr1 = DecisionTreeRegressor()
Fit1 = regr1.fit(X_train, Y_train)
```

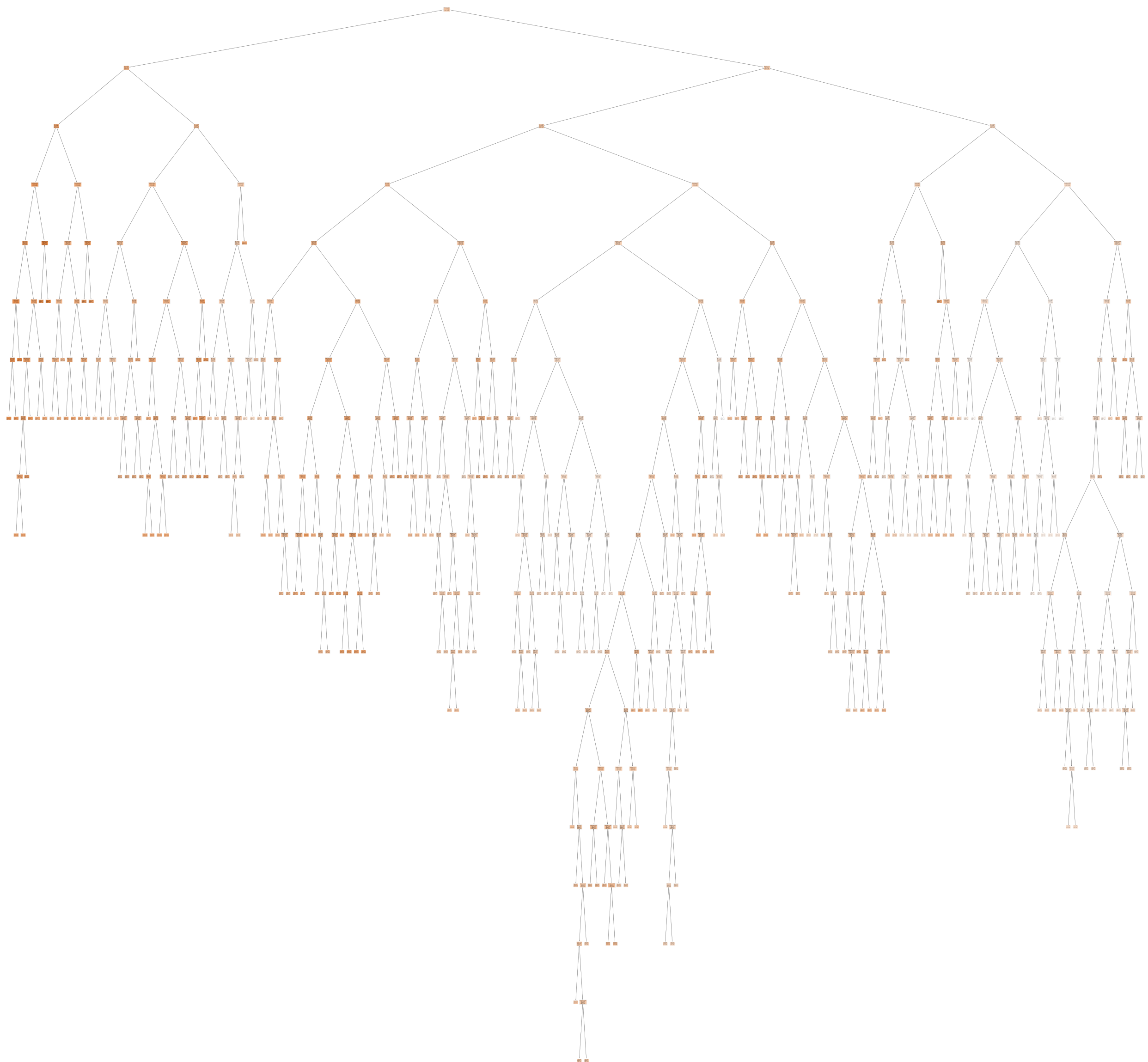
In [6]:
```
pred1 = regr1.predict(X_test)

plt.scatter(pred1, Y_test, label='Sales')
plt.plot([0, 1], [0, 1], '--k', transform=plt.gca().transAxes)
plt.xlabel('pred1')
plt.ylabel('Y_test')

print("Single Tree MSE =",(mean_squared_error(Y_test, pred1)))
```

```
Single Tree MSE = 5.178925833333335
```



The MSE indicated that our predction is off by $5.18 in sales.

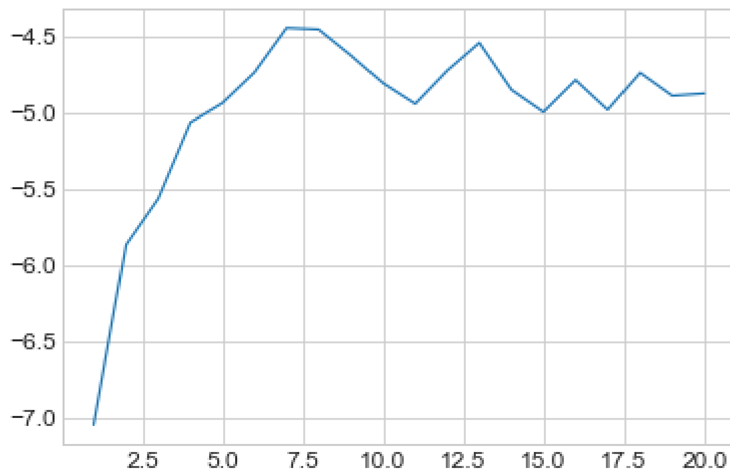
In [7]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (100,100))
tree.plot_tree(regr1,feature_names = X_train.columns,filled = True, class_n
```



After plotting the tree without pruning as indicated by the question, I cannot even read the tree, I will run CV to determine the optimal max_depth.

In [8]:
```
tree_depth = []

for i in range(1,21):
    cv_tree = DecisionTreeRegressor(max_depth=i)
    scores  = cross_val_score(estimator=cv_tree, X=X_train, y=Y_train, cv=5
    tree_depth.append(scores.mean())


plt.plot(range(1,21), tree_depth)
```

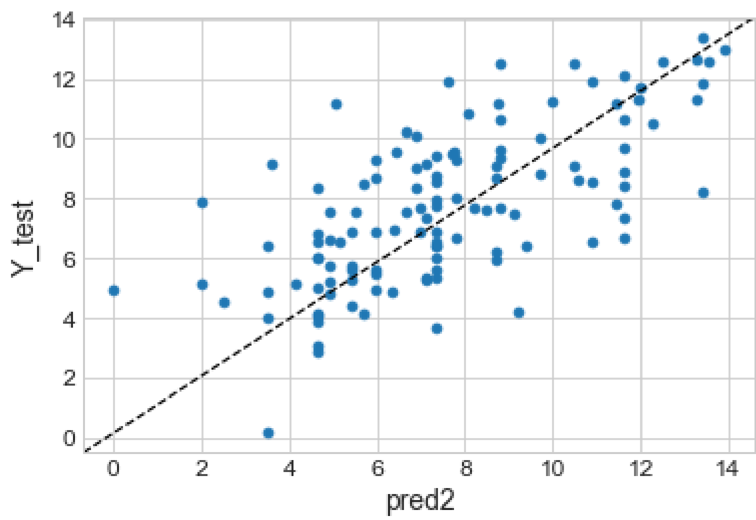
Out[8]:
```
[<matplotlib.lines.Line2D at 0x7f8bd55ce880>]
```



The tree depth with the lowest error is around 8.

In [9]:
```
regr2 = DecisionTreeRegressor(max_depth=8)
Fit2 = regr2.fit(X_train, Y_train)
pred2 = regr2.predict(X_test)
plt.scatter(pred2, Y_test, label='Sales')
plt.plot([0, 1], [0, 1], '--k', transform=plt.gca().transAxes)
plt.xlabel('pred2')
plt.ylabel('Y_test')

print("Tree MSE =",(mean_squared_error(Y_test, pred2)))
```

```
Tree MSE = 5.035580699380617
```



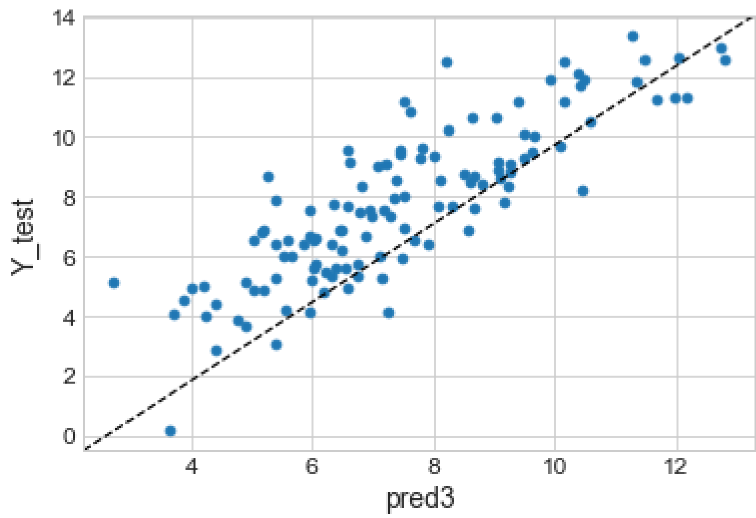
Pruning the tree improved the test MSE.

In [10]:
```
regr3 = RandomForestRegressor(max_features=10, random_state=100) #There are
regr3.fit(X_train, Y_train)
```

Out[10]:
```
RandomForestRegressor(max_features=10, random_state=100)
```

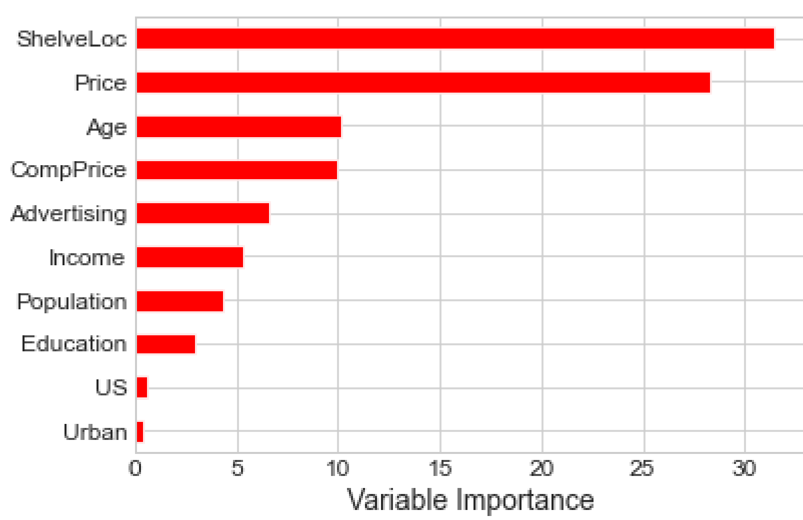
In [11]:
```
pred3 = regr3.predict(X_test)
plt.scatter(pred3, Y_test, label='Sales')
plt.plot([0, 1], [0, 1], '--k', transform=plt.gca().transAxes)
plt.xlabel('pred3')
plt.ylabel('Y_test')
print("Bagging MSE =",(mean_squared_error(Y_test, pred3)))
```

```
Bagging MSE = 2.0138301290833347
```



The bagging approach significantly reduced the test MSE to approximately 2.

In [12]:
```
Importance = pd.DataFrame({'Importance':regr3.feature_importances_*100},ind
Importance.sort_values(by='Importance', axis=0, ascending=True).plot(kind='
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```



Price and Shelveloc are the most important features. Age, CompPrice, Income, Population and Advertising also have an effect, but the rest of the variables seem to have little to no effect on Sales.

I will run a Random Forest to analyze the data, but first I will run a loop to determine the optimal number of features to include in the fit.

number of features to include in the fit.

In [84]:
```
accuracy = []
for i in np.arange(1, 11):
    regr0 = RandomForestRegressor(max_features=i)
    regr0.fit(X_train, Y_train)
    pred = regr0.predict(X_test)
    print(i)
    print("MSE =",(mean_squared_error(Y_test, pred)))
```

```
1
MSE = 3.3247541268333336
2
MSE = 2.5606566258333325
3
MSE = 2.2698978519166646
4
MSE = 2.131636113416667
5
MSE = 2.1351458664166656
6
MSE = 1.9519739814999997
7
MSE = 2.1192106358333342
8
MSE = 2.0536701567500004
9
MSE = 2.193863737000001
10
MSE = 2.178218638833334
```

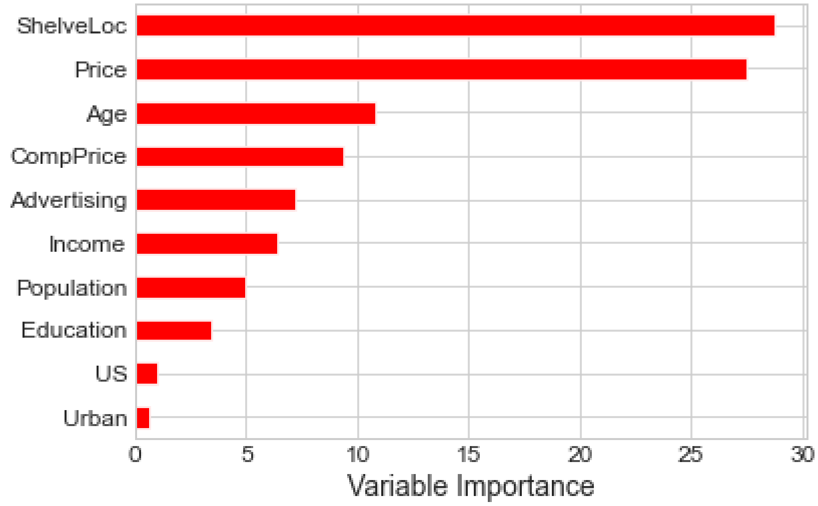
The results indicate that including 6 features would be optimal.

In [85]:
```
regr4 = RandomForestRegressor(max_features=6, random_state=1)
regr4.fit(X_train, Y_train)
pred4 = regr4.predict(X_test)
print("RF MSE =",(mean_squared_error(Y_test, pred4)))
```

```
RF MSE = 1.9921272726666663
```

Random Forest with 6 features achieved the lowest MSE.

In [15]:
```
Importance = pd.DataFrame({'Importance':regr4.feature_importances_*100},ind
Importance.sort_values(by='Importance', axis=0, ascending=True).plot(kind='
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```



Price and Shelveloc are the most important features. Age, CompPrice, Income, Population and Advertising also have an effect, but the rest of the variables seem to have little to no effect on Sales.

8.10

In [16]:
```
data = pd.read_csv('desktop/Hitters.csv')
```

In [17]:
```
data
```

Out[17]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWal
0	293	66	1	30	29	14	1	293	66	1	30	29	
1	315	81	7	24	38	39	14	3449	835	69	321	414	3
2	479	130	18	66	72	76	3	1624	457	63	224	266	2
3	496	141	20	65	78	37	11	5628	1575	225	828	838	3
4	321	87	10	39	42	30	2	396	101	12	48	46	
...	...	...	...	...	...	...	...	...	...	...	...	...	
317	497	127	7	65	48	37	5	2703	806	32	379	311	1
318	492	136	5	76	50	94	12	5511	1511	39	897	451	8
319	475	126	3	61	43	52	6	1700	433	7	217	93	1
320	573	144	9	85	60	78	8	3198	857	97	470	420	3
321	631	170	9	77	44	31	11	4908	1457	30	775	357	2

322 rows × 20 columns

In [18]:
```
data['League'] = data['League'].map({'A':0, 'N':1})
data['Division'] = data['Division'].map({'E':0, 'W':1})
data['NewLeague'] = data['NewLeague'].map({'A':0, 'N':1})
```

In [19]:
```
df4 = data.dropna()
```

In [20]:
```
df4
```

Out[20]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWal
1	315	81	7	24	38	39	14	3449	835	69	321	414	3
2	479	130	18	66	72	76	3	1624	457	63	224	266	2
3	496	141	20	65	78	37	11	5628	1575	225	828	838	3
4	321	87	10	39	42	30	2	396	101	12	48	46	
5	594	169	4	74	51	35	11	4408	1133	19	501	336	1
...	...	...	...	...	...	...	...	...	...	...	...	...	
317	497	127	7	65	48	37	5	2703	806	32	379	311	1
318	492	136	5	76	50	94	12	5511	1511	39	897	451	8
319	475	126	3	61	43	52	6	1700	433	7	217	93	1
320	573	144	9	85	60	78	8	3198	857	97	470	420	3
321	631	170	9	77	44	31	11	4908	1457	30	775	357	2

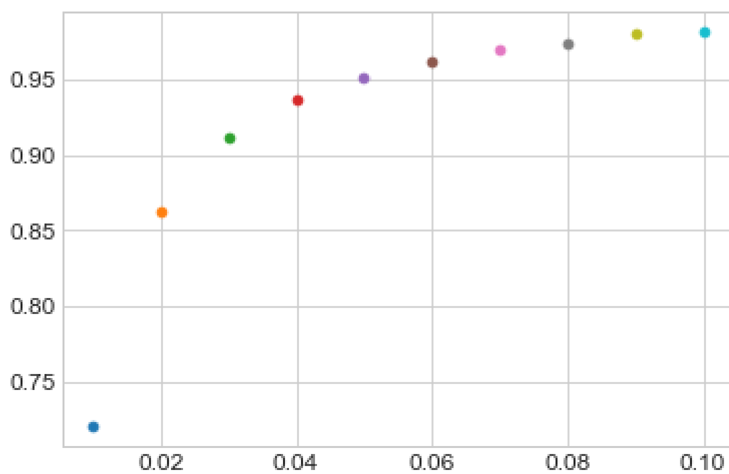
263 rows × 20 columns

In [21]:
```
df4['Salary'] = np.log(df4['Salary'])
```

In [22]:
```
X1 = df4.drop('Salary', axis=1)
Y1 = df4.Salary
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, train_size=
```

In [23]:
```
for i in np.arange(1, 11):
    regr = GradientBoostingRegressor(learning_rate=i/100)
    Fit20 = regr.fit(X1_train, Y1_train)
    plt.scatter(i/100,Fit20.score(X1_train, Y1_train))
    print(i/100)
    print("Score =",(Fit20.score(X1_train, Y1_train)))
```

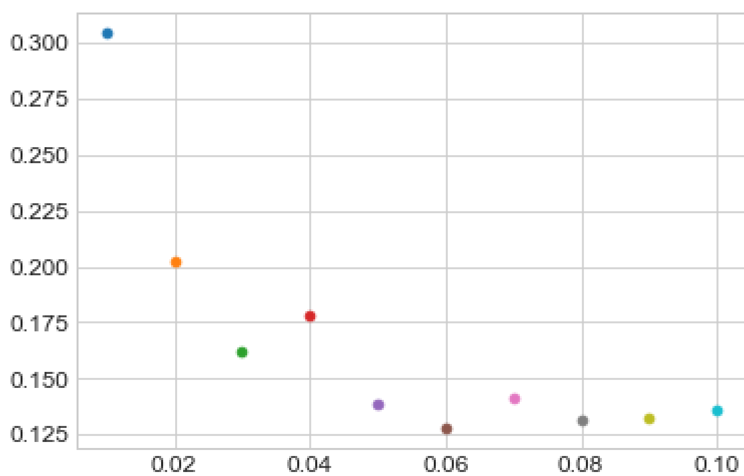
```
0.01
Score = 0.7197430386539123
0.02
Score = 0.8626862873939152
0.03
Score = 0.9121359565824537
0.04
Score = 0.9363616341527955
0.05
Score = 0.9508475306195591
0.06
Score = 0.9615003645168686
0.07
Score = 0.9696744800968778
0.08
Score = 0.9740061429518009
0.09
Score = 0.9799803377762599
0.1
Score = 0.9822427501520995
```



The training score increases as the learning rate increases.

In [83]:
```
for i in np.arange(1, 11):
    regr = GradientBoostingRegressor(learning_rate=i/100)
    regr.fit(X1_train, Y1_train)
    pred = regr.predict(X1_test)
    plt.scatter(i/100,mean_squared_error(Y1_test, pred))
    print(i/100)
    print("MSE =",(mean_squared_error(Y1_test, pred)))
```

```
0.01
MSE = 0.3048293126093497
0.02
MSE = 0.20204658286099078
0.03
MSE = 0.16233217372253966
0.04
MSE = 0.1780180017053089
0.05
MSE = 0.13884219433514353
0.06
MSE = 0.12745306457044206
0.07
MSE = 0.14121920792581444
0.08
MSE = 0.13126521236871347
0.09
MSE = 0.1322173103662674
0.1
MSE = 0.1360423189188568
```



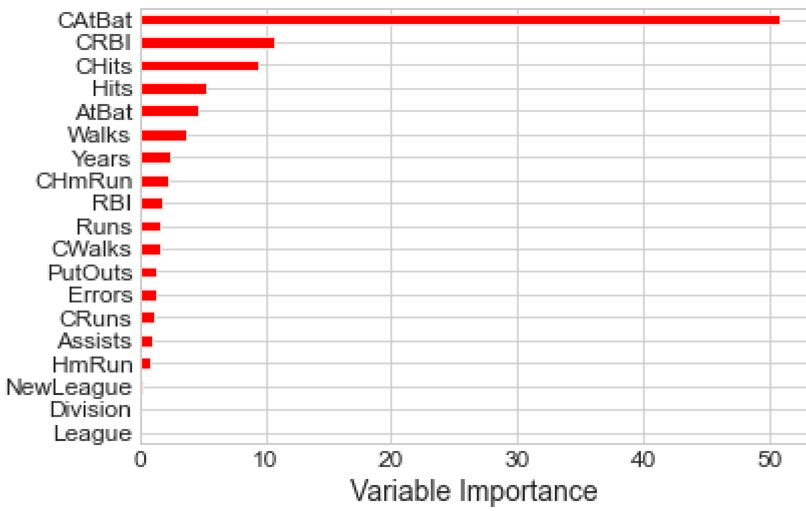
A learning rate of 0.06 results in the lowest test MSE.

In [25]:
```
regr6 = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.06, ra
regr6.fit(X1_train, Y1_train)
print("Boosting MSE =",(mean_squared_error(Y1_test, regr6.predict(X1_test))
```

```
Boosting MSE = 0.14035065773636962
```

Using a 1000 trees and the optimal learning rate, 0.06, boosting produced a low MSE of 0.14.

In [26]:
```
feature_importance = regr6.feature_importances_*100
rel_imp = pd.Series(feature_importance, index=X1.columns).sort_values(inpla
rel_imp.T.plot(kind='barh', color='r', )
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```



CatBat is by the far the most important variable. CRBI and CHITS also seem to have an affect on Salaries, however, the rest of the variables have little to no effect on Salaries.

In [27]:
```
regr7 = RandomForestRegressor(max_features=19, random_state=1) #There are 1
regr7.fit(X1_train, Y1_train)
```

Out[27]:
```
RandomForestRegressor(max_features=19, random_state=1)
```

In [28]:
```
pred7 = regr7.predict(X1_test)
print("Bagging MSE =",(mean_squared_error(Y1_test, pred7)))
```

```
Bagging MSE = 0.14302176494167607
```

The bagging MSE is very close to that of the Boosting MSE, however, the bagging MSE is higher by 0.003, Boosting performed better.

8.12

In [29]:
```
df5 = pd.read_csv('desktop/wagess.csv')
```

In [30]:
```
df5
```

Out[30]:

	wage	education	experience	age	ethnicity	region	gender	occupation	sector	union
0	5.10	8	21	35	hispanic	other	female	worker	manufacturing	no
1	4.95	9	42	57	cauc	other	female	worker	manufacturing	no
2	6.67	12	1	19	cauc	other	male	worker	manufacturing	no
3	4.00	12	4	22	cauc	other	male	worker	other	no
4	7.50	12	17	35	cauc	other	male	worker	other	no
...	...	...	...	...	...	...	...	...	...	...
529	11.36	18	5	29	cauc	other	male	technical	other	no
530	6.10	12	33	51	other	other	female	technical	other	no
531	23.25	17	25	48	other	other	female	technical	other	yes
532	19.88	12	13	31	cauc	south	male	technical	other	yes
533	15.38	16	33	55	cauc	other	male	technical	manufacturing	no

534 rows × 11 columns

In [31]:
```
.f5['ethnicity'] = df5['ethnicity'].map({'cauc':0, 'other':1,'hispanic':2})
.f5['region'] = df5['region'].map({'other':0, 'south':1})
.f5['gender'] = df5['gender'].map({'male':0, 'female':1})
.f5['sector'] = df5['sector'].map({'construction':0, 'manufacturing':1,'othe
.f5['union'] = df5['union'].map({'yes':0, 'no':1})
.f5['married'] = df5['married'].map({'yes':0, 'no':1})
.f5['occupation'] = df5['occupation'].map({'services':0, 'sales':1,'worker':
```

In [32]:
```
X2 = df5.drop('wage', axis=1)
Y2 = df5.wage
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, Y2, train_size=
```

In [81]:
```
for i in np.arange(1, 11):
    regr = RandomForestRegressor(max_features=i)
    regr.fit(X2_train, Y2_train)
    pred9 = regr.predict(X2_test)
    print(i)
    print("MSE =",(mean_squared_error(Y2_test, pred9)))
```

```
1
MSE = 23.34497081980951
2
MSE = 22.922261533544898
3
MSE = 22.777856229302675
4
MSE = 23.60210053963942
5
MSE = 23.253743382329503
6
MSE = 23.709806637781412
7
MSE = 23.758745613871916
8
MSE = 24.414999044408
9
MSE = 23.62765147208681
10
MSE = 24.315375534574603
```

Based on the results, using 3 features to predict wages seems optimal.

In [79]:
```
regr8 = RandomForestRegressor(max_features=3, random_state=1)
regr8.fit(X2_train, Y2_train)

pred10 = regr8.predict(X2_test)
print("RF MSE =",(mean_squared_error(Y2_test, pred10)))
```

```
RF MSE = 23.166513313804177
```

In [35]:
```
regr9 = RandomForestRegressor(max_features=10, random_state=1) #BAGGING
regr9.fit(X2_train, Y2_train)
pred11 = regr9.predict(X2_test)
print("Bagging MSE =",(mean_squared_error(Y2_test, pred11)))
```

```
Bagging MSE = 24.78110127278401
```

In [36]:
```
for i in np.arange(1, 11):
    regr11 = GradientBoostingRegressor(learning_rate=i/100)
    regr11.fit(X2_train, Y2_train)
    pred13 = regr11.predict(X2_test)
    print(i/100)
    print("MSE =",(mean_squared_error(Y2_test, pred13)))
```

```
0.01
MSE = 22.953572478340654
0.02
MSE = 22.508237480819623
0.03
MSE = 22.563564254424158
0.04
MSE = 22.905329008491282
0.05
MSE = 23.10961210241409
0.06
MSE = 23.627362934293817
0.07
MSE = 23.347095667147578
0.08
MSE = 23.814838797037808
0.09
MSE = 23.634478428397557
0.1
MSE = 25.2322341670231
```

Learning rate of 0.02 has the lowest MSE, now I will fit a model using 500 trees and 0.02 learning rate.

In [37]:
```
regr12 = GradientBoostingRegressor(n_estimators=500, learning_rate=0.02, ra
regr12.fit(X2_train, Y2_train)
print("Boosting MSE =",(mean_squared_error(Y2_test, regr12.predict(X2_test)
```

```
Boosting MSE = 24.4812121708686
```

In [38]:
```
mod1 = smf.ols(formula = 'wage ~ education + experience + experience + age
res1 = mod1.fit()
print(res1.summary())
```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:                   wage   R-squared:
0.312
Model:                            OLS   Adj. R-squared:
0.299
Method:                 Least Squares   F-statistic:
23.76
Date:                Wed, 25 May 2022   Prob (F-statistic):           5.8
8e-37
Time:                        19:26:38   Log-Likelihood:                -1
531.3
No. Observations:                 534   AIC:
3085.
Df Residuals:                     523   BIC:
3132.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
Intercept      1.3336      6.652      0.200      0.841    -11.735       1
4.402
education      0.9098      1.086      0.838      0.402     -1.223
3.042
experience     0.3092      1.083      0.285      0.775     -1.819
2.437
age           -0.2184      1.082     -0.202      0.840     -2.345
1.908
ethnicity     -0.4281      0.362     -1.181      0.238     -1.140
0.284
region        -0.7034      0.419     -1.680      0.094     -1.526
0.119
gender        -2.0708      0.386     -5.360      0.000     -2.830       -
1.312
occupation     0.7336      0.141      5.217      0.000      0.457
1.010
sector        -0.4700      0.361     -1.303      0.193     -1.179
0.239
union         -1.5605      0.502     -3.107      0.002     -2.547       -
0.574
married       -0.3220      0.411     -0.784      0.433     -1.129
0.485
==============================================================================
=====
Omnibus:                      257.260   Durbin-Watson:
1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             261
2.084
```

```
Skew:                           1.856   Prob(JB):
0.00
Kurtosis:                      13.180   Cond. No.                        1.6
9e+03
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.69e+03. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [39]:
```
res1.mse_model
```

Out[39]:
```
439.69785003330253
```

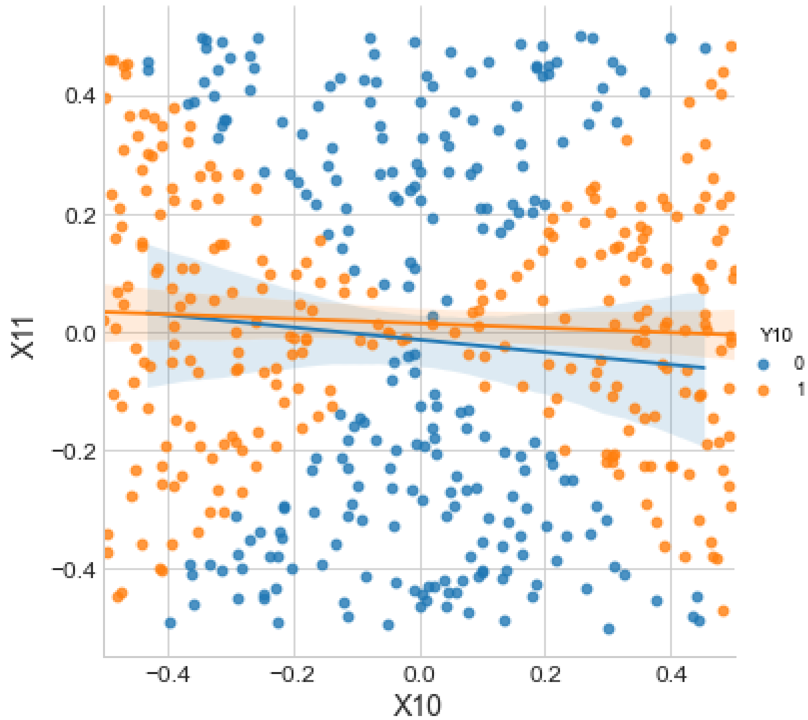
The MSE for the OLS model is significantly higher compared to all other models. Random Forest with max_features = 3 performed best.

9.5

In [40]:
```
p = 2
n=500

np.random.seed(100)

X10 = np.random.uniform(0,1,n)-0.5
X11 = np.random.uniform(0,1,n)-0.5

Y10 = 1*(X10**2 - X11**2 > 0)
df10 = pd.DataFrame({'X10':X10, 'X11':X11, 'Y10':Y10})
```

In [41]:
```
sns.lmplot(x='X10', y='X11', data=df10, fit_reg=True, hue='Y10')
```

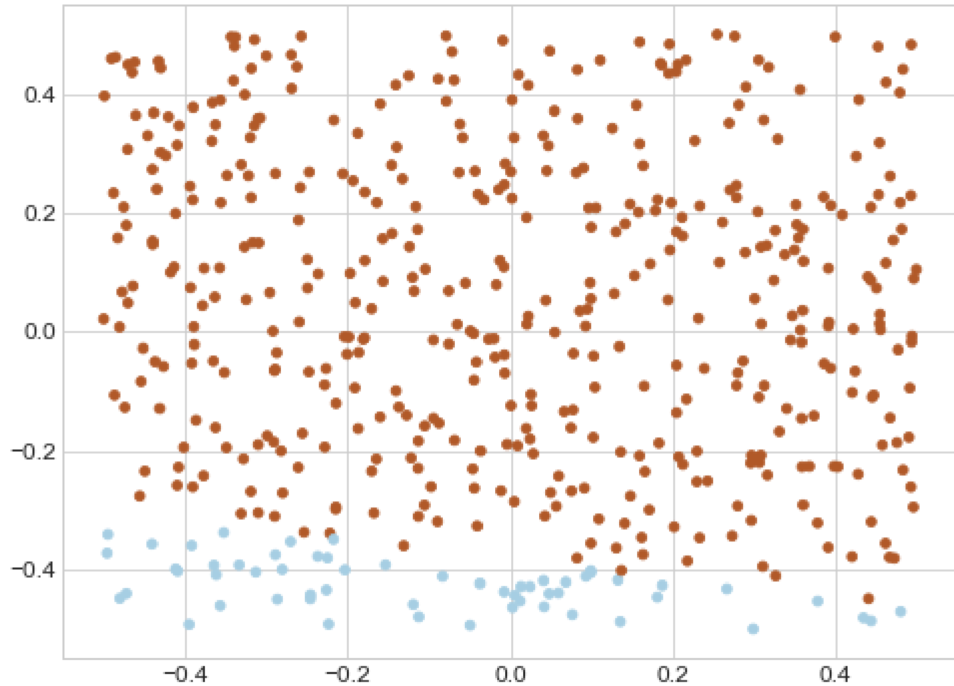
Out[41]:
```
<seaborn.axisgrid.FacetGrid at 0x7f8be161e040>
```



In [42]:
```
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
lr= LogisticRegression()
lr_fit = lr.fit(df10[['X10', 'X11']], df10['Y10'])
lr_pred = lr_fit.predict(df10.drop('Y10', axis =1))
plt.figure(figsize = (8,6))
plt.scatter(df10['X10'], df10['X11'], c = lr_pred, cmap= mpl.cm.Paired)
```

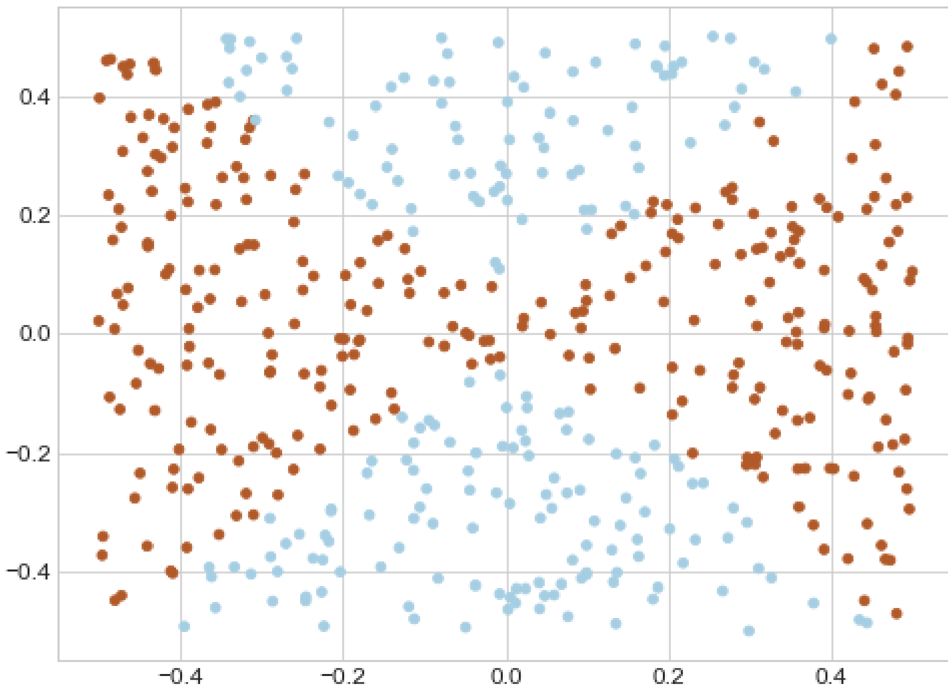
Out[43]: <matplotlib.collections.PathCollection at 0x7f8bbe37a9a0>



In [44]:
```
df10['X12'] = df10['X10']**2
df10['X13'] = df10['X11']**2
```

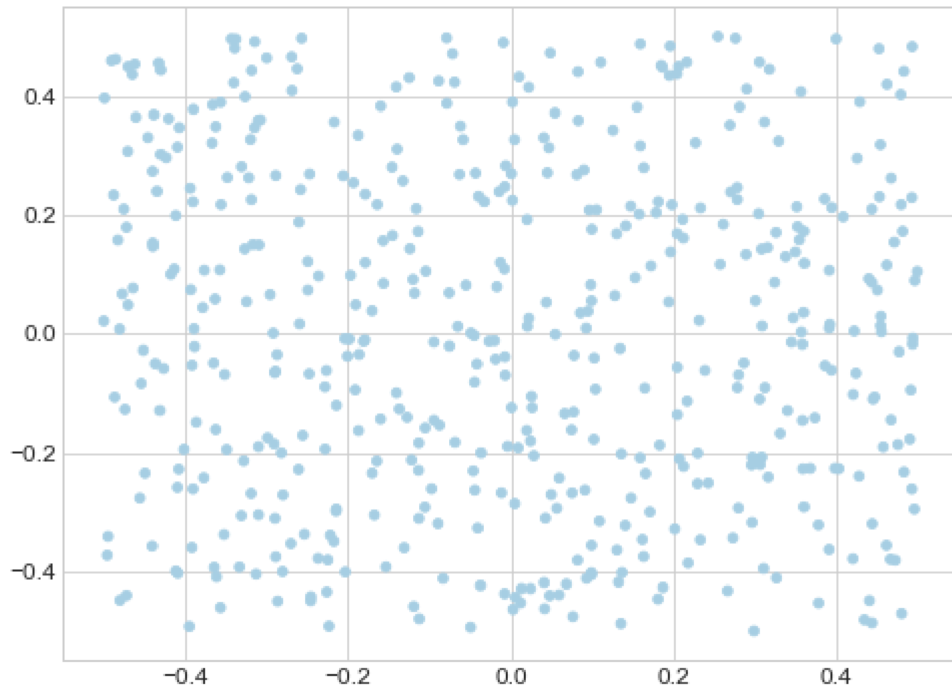
In [45]:
```
lr1 = LogisticRegression()
lr1_fit = lr1.fit(df10.drop('Y10', axis=1), df10['Y10'])
lr1_pred = lr1_fit.predict(df10.drop('Y10', axis=1))
plt.figure(figsize = (8,6))
plt.scatter(df10['X10'], df10['X11'], c = lr1_pred, cmap= mpl.cm.Paired)
```

Out[45]: <matplotlib.collections.PathCollection at 0x7f8bb8fcf760>



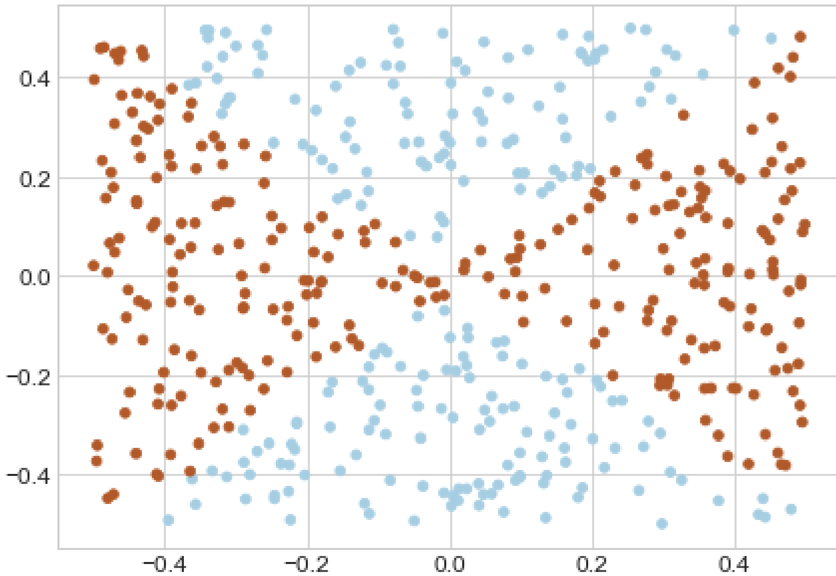
In [86]:
```
svc8 = SVC(C=8, kernel='linear')
svc8.fit(df10.iloc[:,0:2], df10['Y10'])
svc8_pred = svc8.predict(df10.iloc[:,0:2])
plt.figure(figsize = (8,6))
plt.scatter(df10['X10'], df10['X11'], c = svc8_pred, cmap= mpl.cm.Paired)
```

Out[86]: <matplotlib.collections.PathCollection at 0x7f8bbca8a310>



In [47]:
```
svm1 = SVC(C=8, kernel='rbf', degree=2)
svm1.fit(df10.iloc[:,0:2], df10['Y10'])
svm1_pred = svm1.predict(df10.iloc[:,0:2])
plt.figure(figsize = (7,5))
plt.scatter(df10['X10'], df10['X11'], c = svm1_pred, cmap= mpl.cm.Paired)
```

Out[47]: <matplotlib.collections.PathCollection at 0x7f8bbd6832e0>



Observing the plots, the Linear logit performed well separating both classes but linear SVC classified all observations in the same class, while the non-linear Logit and the non-linear SVM produced similar results.

9.7

In [48]:
```
df13 = pd.read_csv('desktop/Auto.csv')
```

In [49]:
```
df13
```

Out[49]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
387	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
388	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
389	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
390	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
391	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

392 rows × 9 columns

In [50]:
```
df14 = df13.drop(axis=1, index=None, columns='name', level=None, inplace=Fa
```

In [51]:
```
df14
```

Out[51]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	18.0	8	307.0	130	3504	12.0	70	1
1	15.0	8	350.0	165	3693	11.5	70	1
2	18.0	8	318.0	150	3436	11.0	70	1
3	16.0	8	304.0	150	3433	12.0	70	1
4	17.0	8	302.0	140	3449	10.5	70	1
...	...	...	...	...	...	...	...	...
387	27.0	4	140.0	86	2790	15.6	82	1
388	44.0	4	97.0	52	2130	24.6	82	2
389	32.0	4	135.0	84	2295	11.6	82	1
390	28.0	4	120.0	79	2625	18.6	82	1
391	31.0	4	119.0	82	2720	19.4	82	1

392 rows × 8 columns

In [52]:
```
df15 = df14.astype('int64')
```

In [53]:
```
df15
```

Out[53]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	18	8	307	130	3504	12	70	1
1	15	8	350	165	3693	11	70	1
2	18	8	318	150	3436	11	70	1
3	16	8	304	150	3433	12	70	1
4	17	8	302	140	3449	10	70	1
...	...	...	...	...	...	...	...	...
387	27	4	140	86	2790	15	82	1
388	44	4	97	52	2130	24	82	2
389	32	4	135	84	2295	11	82	1
390	28	4	120	79	2625	18	82	1
391	31	4	119	82	2720	19	82	1

392 rows × 8 columns

In [54]:
```
df15["mpg01"] = (df15["mpg"] >= df15["mpg"].median()).astype(int)
```

In [55]:
```
df15
```

Out[55]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg01
0	18	8	307	130	3504	12	70	1	0
1	15	8	350	165	3693	11	70	1	0
2	18	8	318	150	3436	11	70	1	0
3	16	8	304	150	3433	12	70	1	0
4	17	8	302	140	3449	10	70	1	0
...	...	...	...	...	...	...	...	...	...
387	27	4	140	86	2790	15	82	1	1
388	44	4	97	52	2130	24	82	2	1
389	32	4	135	84	2295	11	82	1	1
390	28	4	120	79	2625	18	82	1	1
391	31	4	119	82	2720	19	82	1	1

392 rows × 9 columns

In [102]:
```
from sklearn import preprocessing
```

In [56]:
```
X01 = df15.drop(['mpg','mpg01'], axis=1)
Y01 = df15.mpg01
```

In [103]:
```
x = preprocessing.scale(X01)
y = np.ravel(Y01)
```

In [58]:
```
costs = [{'C': [0.01, 0.2, 0.5, 1, 5, 10]}]
svc  = GridSearchCV(SVC(kernel='linear'), costs, cv=10, scoring='accuracy',
svc.fit(X01, Y01)
```

Out[58]:
```
GridSearchCV(cv=10, estimator=SVC(kernel='linear'), n_jobs=-1,
             param_grid=[{'C': [0.01, 0.2, 0.5, 1, 5, 10]}],
             scoring='accuracy')
```

In [59]:
```
Optimal_C = svc.best_estimator_
Optimal_C
```

Out[59]:
```
SVC(C=0.5, kernel='linear')
```

Based on the 10-fold cross-validation, a C of 0.5 achieved the highest accuracy score

In [60]:
```
parameters = [{'C': [0.01, 0.2, 0.5, 1, 5 , 10], 'degree': [2, 4, 6, 8]}]
svm_poly   = GridSearchCV(SVC(kernel='poly'), parameters, cv=10, scoring='a
svm_poly.fit(X01, Y01)
```

Out[60]:
```
GridSearchCV(cv=10, estimator=SVC(kernel='poly'), n_jobs=-1,
             param_grid=[{'C': [0.01, 0.2, 0.5, 1, 5, 10],
                          'degree': [2, 4, 6, 8]}],
             scoring='accuracy')
```

In [61]:
```
Optimal_Parameters  = svm_poly.best_estimator_
Optimal_Parameters
```

Out[61]:
```
SVC(C=10, degree=4, kernel='poly')
```

In [113]:
```
#parameters1 = [{'gamma': [0.01, 0.1]}]
#svm_poly = GridSearchCV(SVC(kernel='poly'), parameters1, cv=5, scoring='ac
#svm_poly.fit(X01, Y01)
```

^ this code has been running for days and its just not executing anything.

In [107]:
```
parameters2 = [{'C': [0.01, 0.2, 0.5, 1, 5 , 10], 'degree': [2, 4, 6, 8],'g
svm_rbf = GridSearchCV(SVC(kernel='rbf'), parameters2, cv=10, scoring='accu
svm_rbf.fit(X01, Y01)
```

Out[107]:
```
GridSearchCV(cv=10, estimator=SVC(), n_jobs=-1,
             param_grid=[{'C': [0.01, 0.2, 0.5, 1, 5, 10],
                          'degree': [2, 4, 6, 8],
                          'gamma': [0.01, 0.1, 1, 2, 4]}],
             scoring='accuracy')
```

In [108]:
```
Optimal_Parameters1  = svm_rbf.best_estimator_
Optimal_Parameters1
```

Out[108]:
```
SVC(C=5, degree=2, gamma=0.01)
```

For the SVM using rbf as the kernel, the 10-fold CV indicates that C=10, degree=2 , and gamma=0.01 are optimal

In [109]:
```
svc_linear = SVC(C=0.5, kernel='linear')
clf1 = svc_linear.fit(X01, Y01)

svm_poly = SVC(C=10, kernel='poly', degree=4)
clf2 = svm_poly.fit(X01, Y01)

svm_rbf = SVC(C=10, kernel='rbf', degree=2, gamma=0.01)
clf3 = svm_rbf.fit(X01, Y01)
```

In [110]:
```
rror_name = ['SVC', 'SVM poly', 'SVM radial']
rror_rate = [1-Optimal_C.score(X01, Y01), 1-Optimal_Parameters.score(X01, Y(
rint(pd.DataFrame({'training error rate': error_rate}, index=error_name))
```

```
            training error rate
SVC                    0.081633
SVM poly               0.112245
SVM radial             0.000000
```

SVM with a radial kernel, C=10, and gamma of 0.01 achieved the lowest error.

In [111]:
```
fig  = plt.figure()

fig.suptitle('decison surface using projected features')
labels = ['SVC', 'SVM poly', 'SVM radial' ]
gs = gridspec.GridSpec(3, 1)
for clf, lab, grd in zip([clf1, clf2, clf3], labels, ([0,0], [1,0],[2,0])):
    clf.fit(np.stack((x[:,0], x[:,1]), axis=-1), y)
    ax  = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=np.stack((x[:,0], x[:,1]), axis=-1), y=y,
    plt.title(lab)
plt.show()
```