Logistic Regression Classifier

In [1]:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns

```

Dataset

In [2]:

```
dta = sm.datasets.fair.load_pandas().data
dta.head(5)
```

Out[2]:

	rate_marriage	age	yrs_married	children	religious	educ	occupation	occupation_husb	
0	3.0	32.0	9.0	3.0	3.0	17.0	2.0	5.0	0
1	3.0	27.0	13.0	3.0	1.0	14.0	3.0	4.0	3.
2	4.0	22.0	2.5	0.0	1.0	16.0	3.0	5.0	1.
3	4.0	37.0	16.5	4.0	3.0	16.0	5.0	5.0	0.
4	5.0	27.0	9.0	1.0	1.0	14.0	3.0	4.0	4.

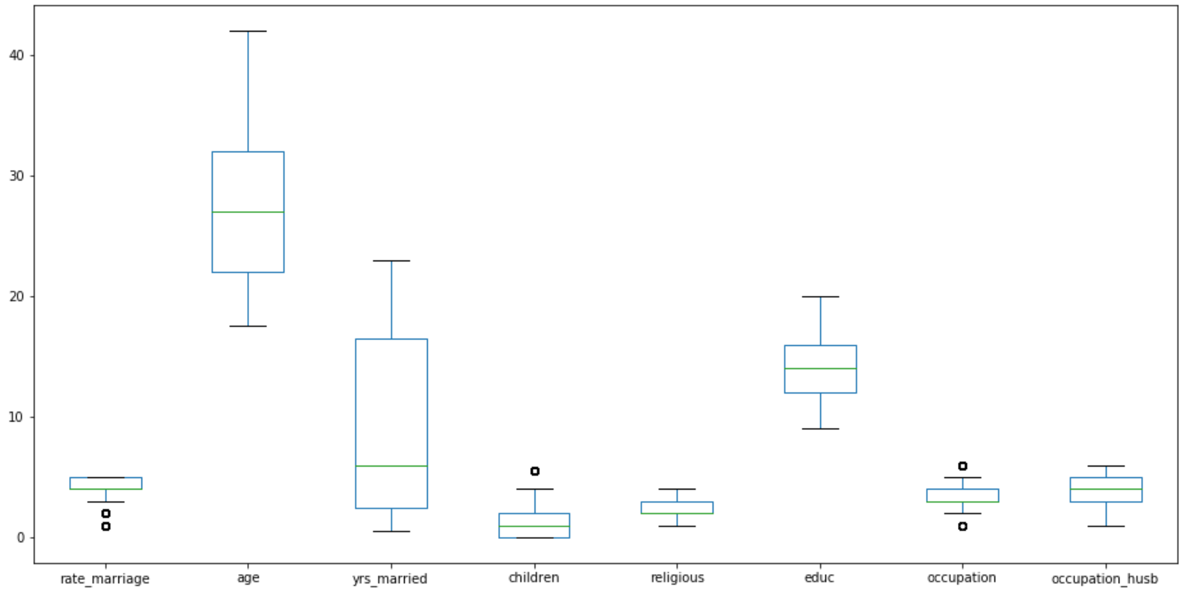
Checking for outliers using box plot

In [3]:

```
dta.drop(['affairs'],axis=1).plot(kind='box',figsize=(16,8))
```

Out[3]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x187078d18b0>
```



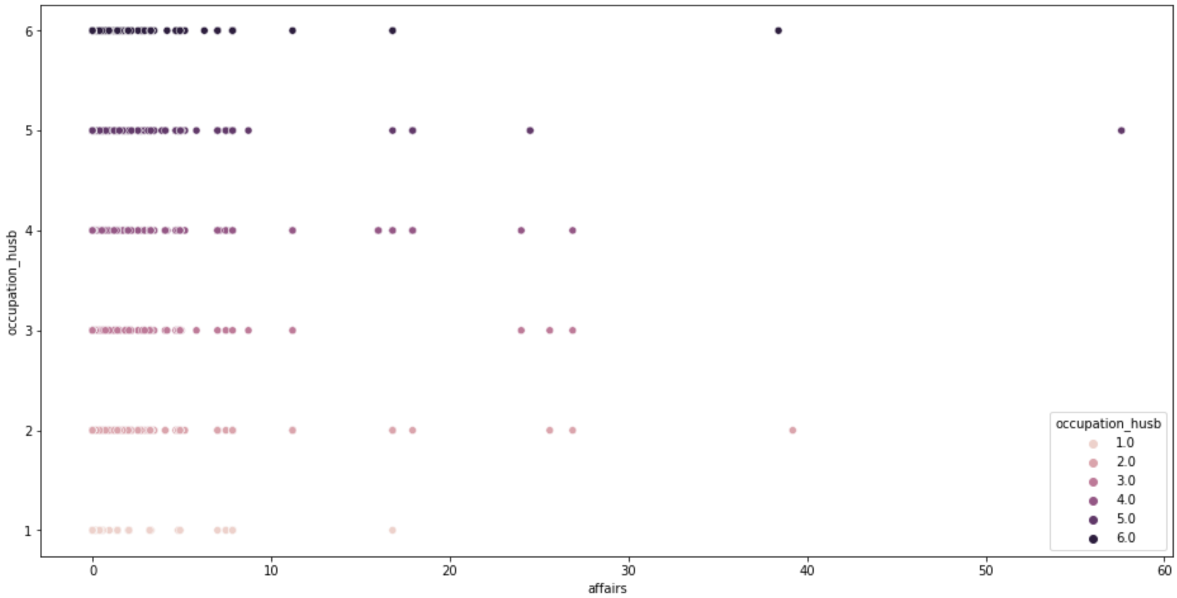
Relation between Affair and Occupation of Husband

In [4]:

```
fig = plt.figure(figsize=(16,8))
sns.scatterplot(x=dta['affairs'],y=dta['occupation_husb'],hue=dta['occupation_husb'])
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18765fea4c0>
```



We can see that the chances of Divorce differes with the occupation hunbands is in.

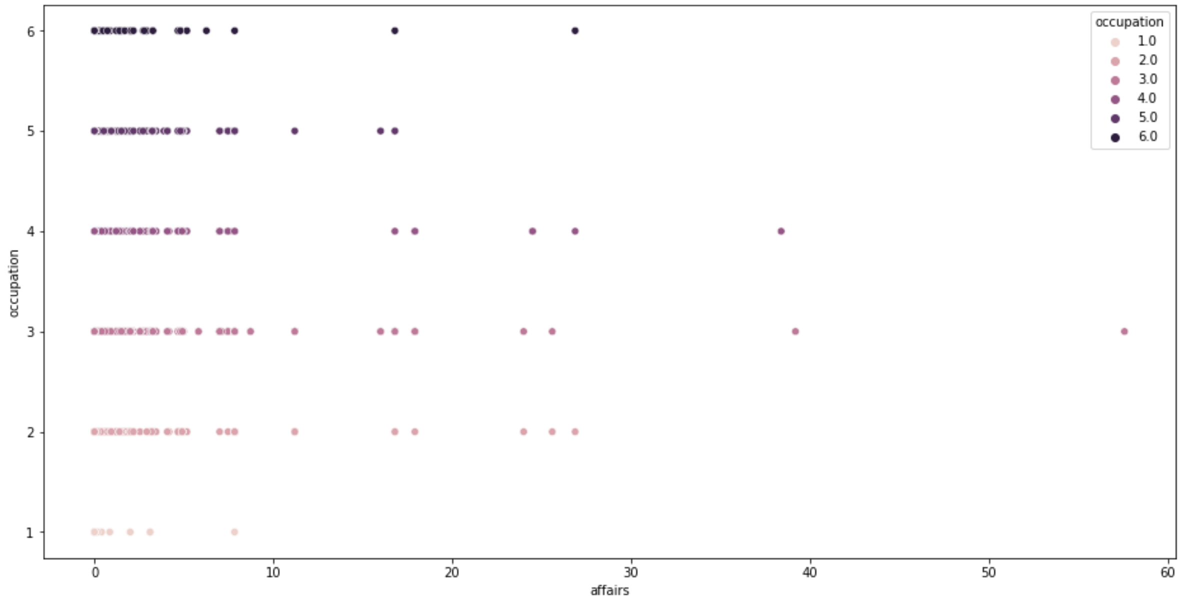
Relation between Affair and Occupation of Women.

In [5]:

```
fig = plt.figure(figsize=(16,8))
sns.scatterplot(x=dta['affairs'],y=dta['occupation'],hue=dta['occupation'])
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18707a698e0>
```



Women having White Collar Job show high chances of Affair.

Information about Dataset

The dataset contains 6366 observations of 9 variables:

> rate_marriage: woman's rating of her marriage
>
> > (1 = very poor, 5 = very good)

> age: woman's age

> yrs_married: number of years married

> children: number of children

> religious: woman's rating of how religious she is
>
> > (1 = not religious, 4 =strongly religious)

> educ: level of education
>
> > (9 = grade school, 12 = high school, 14 = some college, 16 = college graduate, 17 = some graduate school, 20 = advanced degree)

> occupation: woman's occupation
>
> > (1 = student, 2 = farming/semi- skilled/unskilled, 3 = "white collar", 4 = teacher/nurse/writer/technician/skilled, 5 = managerial/business, 6 = professional with advanced degree)

> occupation_husb: husband's occupation (same coding as above)
>
> > (1 = student, 2 = farming/semi- skilled/unskilled, 3 = "white collar", 4 = teacher/nurse/writer/technician/skilled, 5 = managerial/business, 6 = professional with advanced degree)

> affairs: time spent in extra-marital affairs

Making a new Column that will contain Affairs in Binary Form

> 0 - No Affairs

> 1 - Having Affairs

In [6]:

```
dta['Affairs'] = (dta.affairs> 0).astype(int)
dta.head(3)
```

Out[6]:

	rate_marriage	age	yrs_married	children	religious	educ	occupation	occupation_husb	
0	3.0	32.0	9.0	3.0	3.0	17.0	2.0	5.0	0
1	3.0	27.0	13.0	3.0	1.0	14.0	3.0	4.0	3.
2	4.0	22.0	2.5	0.0	1.0	16.0	3.0	5.0	1.

Checking For Null Values

In [7]:

```
print(dta.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6366 entries, 0 to 6365
Data columns (total 10 columns):
rate_marriage      6366 non-null float64
age                6366 non-null float64
yrs_married        6366 non-null float64
children           6366 non-null float64
religious          6366 non-null float64
educ               6366 non-null float64
occupation         6366 non-null float64
occupation_husb    6366 non-null float64
affairs            6366 non-null float64
Affairs            6366 non-null int32
dtypes: float64(9), int32(1)
memory usage: 472.6 KB
None
```

No Null values are there in any column. Data type is Float, with affairs as int

Creating Design Matrix

We are creating two design matrices. The first is the martix of endogenous variables(dependent variables ~ 'X'). The second design matrix is of exogenous variables(Independent Variables ~ 'y'). For this we are uisng dmatrices function from patsy

In [8]:

```
from patsy import dmatrices
y,X = dmatrices('Affairs ~ rate_marriage+age+yrs_married+children+religious+educ+C(occu
X.head(2)
```

Out[8]:

	Intercept	C(occupation)[T.2.0]	C(occupation)[T.3.0]	C(occupation)[T.4.0]	C(occupation)[T.5.0]	C(occupation)[T.6.0]	C(occup
0	1.0	1.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	1.0	0.0	0.0	0.0	

dmatrices returns a dataframe constituting of categories for categorical variables in our case occupation(female occupation) and occupation_husb(Husbands occupations).

It add a contant variable "intercept" to the dataframe.

In [9]:

```
y.head(2)
```

Out[9]:

	Affairs
0	1.0
1	1.0

Converting y into a 1 D flattern array.

In [10]:

```
y = np.ravel(y)
y
```

Out[10]:

```
array([1., 1., 1., ..., 0., 0., 0.])
```

Renaming Columns of categorical variables in X

for simplicity

In [11]:

```
X.rename(inplace=True,columns={'C(occupation)[T.2.0]':'occ_2.0','C(occupation)[T.3.0]':
X.head(5)
```

Out[11]:

	Intercept	occ_2.0	occ_3.0	occ_4.0	occ_5.0	occ_6.0	occ_husb_2.0	occ_husb_3.0	occ
0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
2	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
4	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	

In [12]:

```
print("Info of X:\n")
X.info()
```

```
Info of X:

<class 'pandas.core.frame.DataFrame'>
Int64Index: 6366 entries, 0 to 6365
Data columns (total 17 columns):
Intercept        6366 non-null float64
occ_2.0          6366 non-null float64
occ_3.0          6366 non-null float64
occ_4.0          6366 non-null float64
occ_5.0          6366 non-null float64
occ_6.0          6366 non-null float64
occ_husb_2.0     6366 non-null float64
occ_husb_3.0     6366 non-null float64
occ_husb_4.0     6366 non-null float64
occ_husb_5.0     6366 non-null float64
occ_husb_6.0     6366 non-null float64
rate_marriage    6366 non-null float64
age              6366 non-null float64
yrs_married      6366 non-null float64
children         6366 non-null float64
religious        6366 non-null float64
educ             6366 non-null float64
dtypes: float64(17)
memory usage: 895.2 KB
```

Everything looks good. Now let's check whether anyvalues is null values

In [13]:

```
X.isna().sum()
```

Out[13]:

```
Intercept        0
occ_2.0          0
occ_3.0          0
occ_4.0          0
occ_5.0          0
occ_6.0          0
occ_husb_2.0     0
occ_husb_3.0     0
occ_husb_4.0     0
occ_husb_5.0     0
occ_husb_6.0     0
rate_marriage    0
age              0
yrs_married      0
children         0
religious        0
educ             0
dtype: int64
```

No Null values are there so Moving forward with our next step.

Checking Correlation

Correlation

Relation of a variable with other variables.

In [14]:

```
X.corr()
```

Out[14]:

	Intercept	occ_2.0	occ_3.0	occ_4.0	occ_5.0	occ_6.0	occ_husb_2.0	o
Intercept	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
occ_2.0	NaN	1.000000	-0.348075	-0.251243	-0.143237	-0.052128	0.183782	
occ_3.0	NaN	-0.348075	1.000000	-0.560645	-0.319631	-0.116322	-0.000638	
occ_4.0	NaN	-0.251243	-0.560645	1.000000	-0.230712	-0.083962	-0.083123	
occ_5.0	NaN	-0.143237	-0.319631	-0.230712	1.000000	-0.047868	-0.053426	
occ_6.0	NaN	-0.052128	-0.116322	-0.083962	-0.047868	1.000000	-0.046140	
occ_husb_2.0	NaN	0.183782	-0.000638	-0.083123	-0.053426	-0.046140	1.000000	
occ_husb_3.0	NaN	-0.020904	0.090043	-0.043159	-0.044053	-0.029028	-0.146849	
occ_husb_4.0	NaN	-0.009786	0.011248	0.037341	-0.039932	-0.043541	-0.347951	
occ_husb_5.0	NaN	-0.093292	0.003021	-0.001946	0.114903	-0.030926	-0.316693	
occ_husb_6.0	NaN	-0.059107	-0.101673	0.085766	0.006016	0.218824	-0.153248	
rate_marriage	NaN	-0.019697	-0.053082	0.068882	-0.002109	0.008878	-0.038992	
age	NaN	-0.034223	-0.066371	0.040982	0.079533	0.030676	-0.057368	
yrs_married	NaN	0.004668	-0.021261	-0.026816	0.076820	-0.004912	-0.033451	
children	NaN	0.081182	-0.063298	-0.003235	0.033274	-0.026830	0.001190	
religious	NaN	-0.013129	-0.034986	0.043996	0.004260	0.011784	0.009990	
educ	NaN	-0.217719	-0.335615	0.477505	-0.022121	0.226920	-0.160756	

VIF Score

VIF Score > 5 Indicates sever correlation.

In [15]:

```
# VIF Score
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
from sklearn.preprocessing import StandardScaler as StdS
scaler = StdS()
X_data = scaler.fit_transform(X=X)
vif_score = [vif(X_data,i) for i in range(X_data.shape[1])]
for i in range(len(X.columns)):
    print(X.columns[i],":",vif_score[i])
datset1 = X
```

```
Intercept : nan
occ_2.0 : 19.340780081786384
occ_3.0 : 39.33561800583917
occ_4.0 : 32.93190959613787
occ_5.0 : 17.05716507563386
occ_6.0 : 3.697958521454455
occ_husb_2.0 : 5.5662915598555065
occ_husb_3.0 : 2.9910696353751987
occ_husb_4.0 : 6.930281429175677
occ_husb_5.0 : 6.577077423174822
occ_husb_6.0 : 3.1852660411536395
rate_marriage : 1.0387458559035507
age : 5.477889737628554
yrs_married : 7.16961101396522
children : 2.5856914303595504
religious : 1.0375560897654312
educ : 1.6357895734289674

C:\Users\adity\anaconda3\lib\site-packages\statsmodels\regression\linear_mod
el.py:1717: RuntimeWarning: invalid value encountered in double_scalars
  return 1 - self.ssr/self.uncentered_tss
```

In [16]:

```
for i in range(len(X.columns)):
    print(X.columns[i],":",vif_score[i])
```

```
Intercept : nan
occ_2.0 : 19.340780081786384
occ_3.0 : 39.33561800583917
occ_4.0 : 32.93190959613787
occ_5.0 : 17.05716507563386
occ_6.0 : 3.697958521454455
occ_husb_2.0 : 5.5662915598555065
occ_husb_3.0 : 2.9910696353751987
occ_husb_4.0 : 6.930281429175677
occ_husb_5.0 : 6.577077423174822
occ_husb_6.0 : 3.1852660411536395
rate_marriage : 1.0387458559035507
age : 5.477889737628554
yrs_married : 7.16961101396522
children : 2.5856914303595504
religious : 1.0375560897654312
educ : 1.6357895734289674
```

Now from this VIF Score it is clear that there are features having very high correlation values. occ_2.0,occ_3.0,occ_4.0 and occ_5.0. In occ_husb - occ_husb_2.0, occ_husb_4.0, occ_husb_5.0 are also having high correlation.

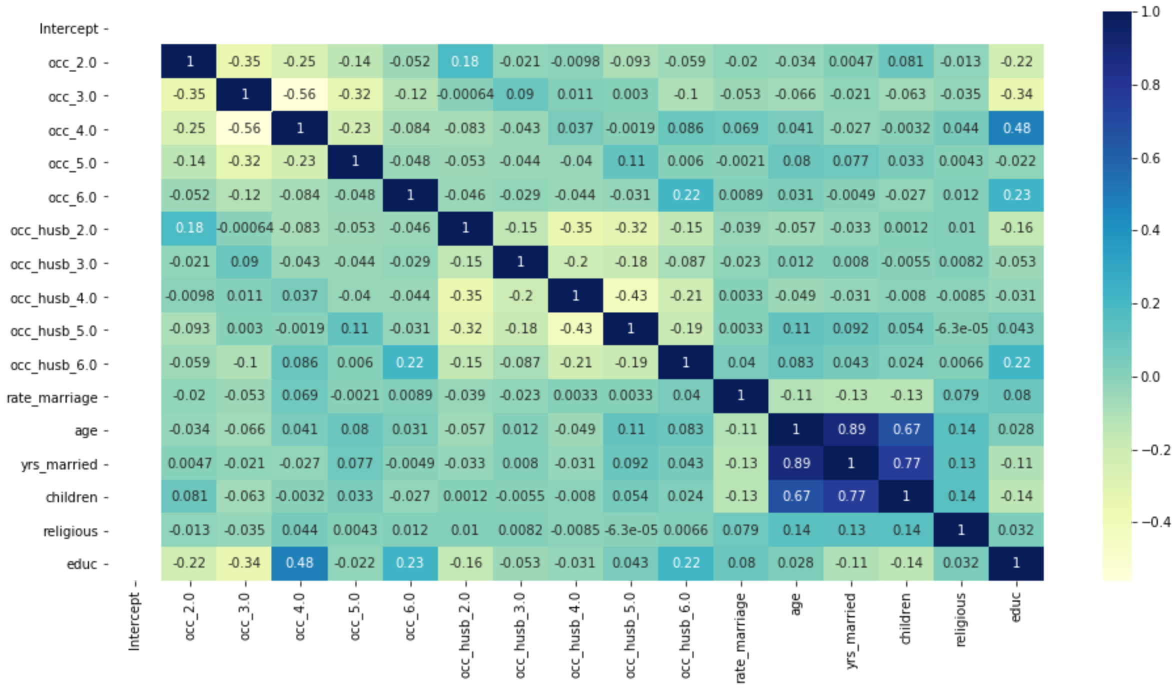
Ploting heat map with correlation value with X.

In [17]:

```
fig = plt.figure(figsize=(16,8))
sns.heatmap(data=X.corr(),cmap="YlGnBu", annot=True)
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1870a279280>
```



As we can observe that in occ_2.0,occ_3.0,occ_4.0 and occ_5.0 .

occ_2.0 with {occ_2.0,occ_3.0,occ_4.0 and occ_5.0} is {1.000000 -0.348075 -0.251243 -0.143237} respectively.

occ_3.0 with {occ_2.0,occ_3.0,occ_4.0 and occ_5.0} is {-0.348075 1.000000 -0.560645 -0.319631} respectively.

occ_4.0 with {occ_2.0,occ_3.0,occ_4.0 and occ_5.0} is {-0.251243 -0.560645 1.000000 -0.230712} respectively.

occ_5.0 with {occ_2.0,occ_3.0,occ_4.0 and occ_5.0} is {-0.143237 -0.319631 -0.230712 1.000000} respectively.

The feature occ_3.0 is having high correlation in respect to all other features. So removing occ_3.0 and lets check the effect on VIF Score.

In [18]:

```
X_data = scaler.fit_transform(X=X.drop('occ_3.0',axis=1))
vif_score = [vif(X_data,i) for i in range(X_data.shape[1])]
for i in range(len(X.drop('occ_3.0',axis=1).columns)):
    print(X.drop('occ_3.0',axis=1).columns[i],":",vif_score[i])
```

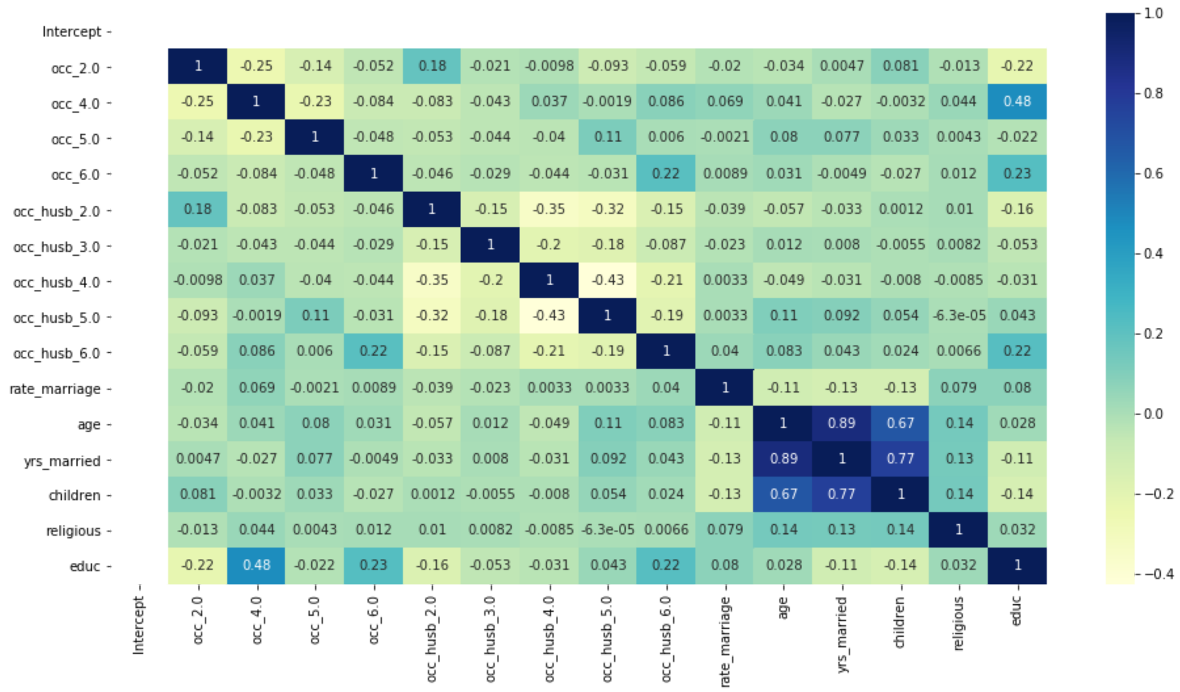
```
Intercept : nan
occ_2.0 : 1.177577731855246
occ_4.0 : 1.58134483216193
occ_5.0 : 1.1518453049129829
occ_6.0 : 1.166552590194721
occ_husb_2.0 : 5.530740700962974
occ_husb_3.0 : 2.9761967717120825
occ_husb_4.0 : 6.8868032112916735
occ_husb_5.0 : 6.533261649523526
occ_husb_6.0 : 3.1752920128006568
rate_marriage : 1.0385349926O2799
age : 5.474644652197843
yrs_married : 7.169399081318488
children : 2.5846493806759088
religious : 1.037500793190026
educ : 1.6292132746346843

C:\Users\adity\anaconda3\lib\site-packages\statsmodels\regression\linear_mod
el.py:1717: RuntimeWarning: invalid value encountered in double_scalars
  return 1 - self.ssr/self.uncentered_tss
```

In [19]:

```
fig = plt.figure(figsize=(16,8))
sns.heatmap(data=X.drop('occ_3.0',axis=1).corr(),cmap="YlGnBu", annot=True)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1870b800070>
```



In [20]:

```
X.head(2)
```

Out[20]:

	Intercept	occ_2.0	occ_3.0	occ_4.0	occ_5.0	occ_6.0	occ_husb_2.0	occ_husb_3.0	occ_hu
0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	

In [21]:

```
from sklearn.model_selection import train_test_split as tts
x_train,x_test,y_train,y_test = tts(X,y,test_size=0.20,random_state=225)
```

Using Stochastic Gradient Descent

In [22]:

```
from sklearn.linear_model import SGDClassifier
model1 = SGDClassifier()
model1.fit(x_train,y_train)
```

Out[22]:

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=Tru
e,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Testing Score

In [23]:

```
model1.score(x_train,y_train)
```

Out[23]:

```
0.43676355066771405
```

Traning Score

In [24]:

```
model1.score(x_test,y_test)
```

Out[24]:

```
0.4340659340659341
```

Using LogisticRegression Classifier

In [25]:

```
from sklearn.linear_model import LogisticRegression
model2 = LogisticRegression(solver='liblinear')
model2.fit(x_train,y_train)
```

Out[25]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbos
e=0,
                   warm_start=False)
```

Traning Score

In [26]:

```
model2.score(x_train,y_train)*100
```

Out[26]:

```
73.19324430479183
```

Testing Score

In [27]:

```
model2.score(x_test,y_test)*100
```

Out[27]:

```
69.78021978021978
```

Evaluation of Classification Model

1. Accuracy
2. Recall
3. Precision
4. F1 Score
5. Specifity
6. AUC (Area Under Curve)
7. RUC(Receiver Operator Characteristic)

Cross Validation

In [28]:

```
# Performing cross validation for model1 and Model 2

from sklearn.model_selection import cross_val_predict
model1_y_predict = cross_val_predict(model1,x_train,y_train,cv=3)
model2_y_predict = cross_val_predict(model2,x_train,y_train,cv=3)
```

Confusion Matrix

--	Actual	Values
Predicted	True Positive	False Positive
Values	False Negative	True Negative

For Model 1

In [29]:

```
from sklearn.metrics import confusion_matrix,recall_score, precision_score, f1_score, a
conf_matrix_model1 = confusion_matrix(y_train,model1_y_predict)
conf_matrix_model1
```

Out[29]:

```
array([[3021,  427],
       [1088,  556]], dtype=int64)
```

For Model 2

In [30]:

```
conf_matrix_model2 = confusion_matrix(y_train,model2_y_predict)
conf_matrix_model2
```

Out[30]:

```
array([[3099,  349],
       [1030,  614]], dtype=int64)
```

In [31]:

```
# Accuracy Recall Precision F1 Score Specifity calculator.
"""
def calulate(c_matrix):
    tp = c_matrix[0][0]
    fp = c_matrix[0][1]
    fn = c_matrix[1][0]
    tn = c_matrix[1][1]
    acc = (tp+tn)/(tp+fp+fn+tn)
    rell = tp/(tp+fn)
    pre = tp/(tp+fp)
    f1 = 2*(pre*rell)/(pre+rell)
    spe = tn/(tn+fp)
    return acc,rell,pre,f1,spe
"""

def get_values(model):
    recall = recall_score(y_train,model)
    precision = precision_score(y_train,model)
    f1 = f1_score(y_train,model)
    accuracy = accuracy_score(y_train,model)
    return print(f'recall: {recall}\nprecision: {precision}\nf1: {f1}\naccuracy: {accur

```

"""m1_accuracy,m1_recall,m1_precision,m1_f1_score,m1_specificity = calulate(conf_matrix_model1) print(f'Confusion Matrix:\n{conf_matrix_model1}\nm1_accuracy: {m1_accuracy}\nm1_recall: {m1_recall}\nm1_precision: {m1_precision}\nm1_f1_score: {m1_f1_score}\nm1_specificity: {m1_specificity}')"""

"""m2_accuracy,m2_recall,m2_precision,m2_f1_score,m2_specificity = calulate(conf_matrix_model2) print(f'Confusion Matrix:{conf_matrix_model2}\nm2_accuracy: {m2_accuracy}\nm2_recall: {m2_recall}\nm2_precision: {m2_precision}\nm2_f1_score: {m2_f1_score}\nm2_specificity: {m2_specificity}')"""

In [32]:

```
print("model 1")
get_values(model1_y_predict)
print("\nModel 2")
get_values(model2_y_predict)
```

```
model 1
recall: 0.3381995133819951
precision: 0.5656154628687691
f1: 0.4232965359725923
accuracy: 0.7024744697564808

Model 2
recall: 0.3734793187347932
precision: 0.6375908618899273
f1: 0.47103950901419256
accuracy: 0.7291830322073841
```

Now each of these values signifies. Let's find out

Accuracy

Total class values accurately classified by the model. Model 1 Accuracy is about 53 % Model 2 Accuracy is about 72 %

Model 2 is more accurate but in the case of classification only model accuracy is not the cirteria of getting an actual accurate model.

Moving forward let's see recall and presision of the model.

Recall

In [33]:

```
print("model 1")
get_values(model1_y_predict)
print("Model 2")
get_values(model2_y_predict)
```

```
model 1
recall: 0.3381995133819951
precision: 0.5656154628687691
f1: 0.4232965359725923
accuracy: 0.7024744697564808
Model 2
recall: 0.3734793187347932
precision: 0.6375908618899273
f1: 0.47103950901419256
accuracy: 0.7291830322073841
```

Recall is defined as the ability of ML Model to correctly predict positive out of all positive results. Taking the recall into consideration we can choose Model 1 .

Precision

It measure of amongst all the positive predictions how many of them were were actaully positive.

Precision of Model 1 is 37% and of Model 2 is 63% . So if we want that our model must accurately predict the outcome we should consider high precision. taking this into consideration we should go with Model 2.

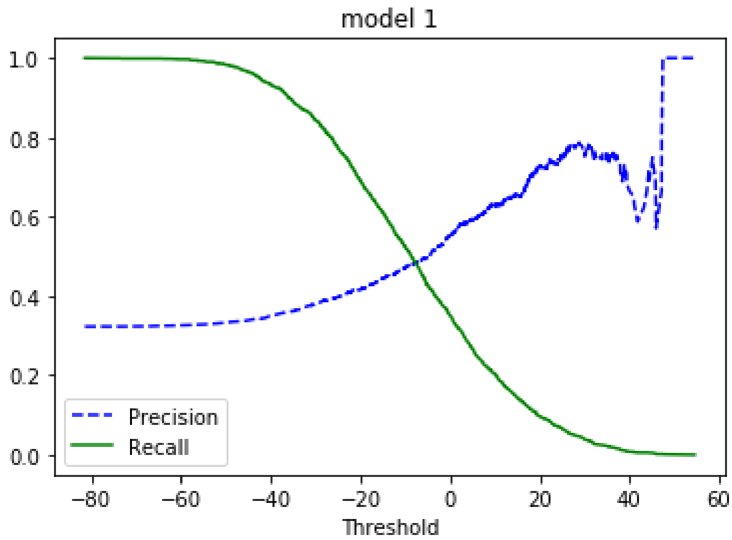
Ploting Precision Recall Curve

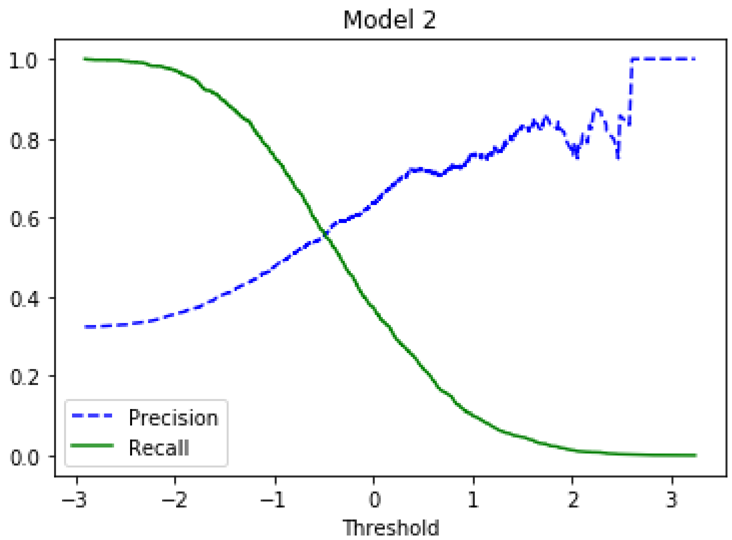
In [34]:

```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds,model):
    plt.title(model)
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.xlabel('Threshold')
    plt.legend()

```

In [35]:

```python
from sklearn.metrics import precision_recall_curve as prc
y_score_m1 = cross_val_predict(model1,x_train,y_train,cv=3,method='decision_function')
y_score_m2 = cross_val_predict(model2,x_train,y_train,cv=3,method='decision_function')
pre_m1, rec_m1 , threshol_m1 = prc(y_train,y_score_m1)
pre_m2, rec_m2 , threshol_m2 = prc(y_train,y_score_m2)


plot_precision_recall_vs_threshold(pre_m1, rec_m1 , threshol_m1,"model 1")
plt.show()


plot_precision_recall_vs_threshold(pre_m2, rec_m2 , threshol_m2,"Model 2")
plt.show()

```





We can oberve with change in threshold we can change the precision and recall of both of our model. But for the time being let's go with the default thresold score.

F1 Score

It's like a trade off factor . classifier having high precision and recall value will get high f1 score. Considering this we should go with Model 1.

ROC

ROC curve is grapgh of True Positive rate Vs false Positive rate True Positive Rate is Recall/Sensitivity while False Positive rate is out of total false prediction how many were actually false.

AUC

Aea under the curve, According to it the model is best which encloses maximum area.

In [36]:

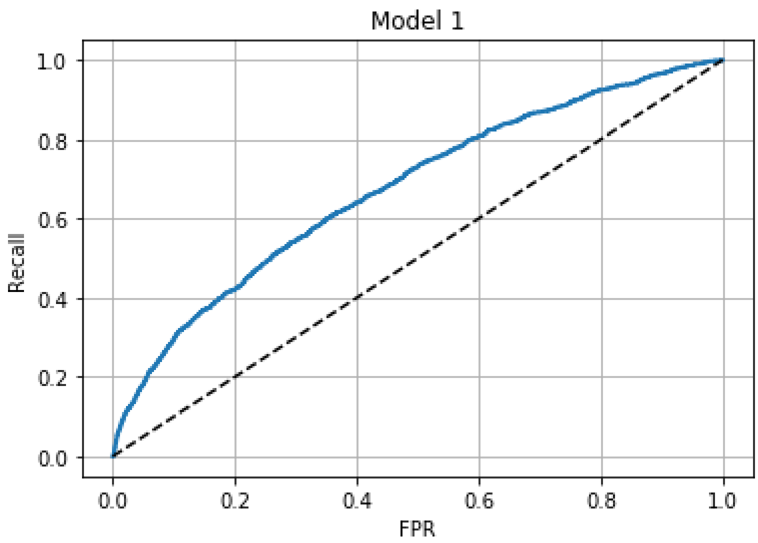
```
from sklearn.metrics import roc_auc_score

roc_auc_m1 = roc_auc_score(y_train,y_score_m1)
roc_auc_m2 = roc_auc_score(y_train,y_score_m2)

print(f'roc_auc_m1: {roc_auc_m1}\nroc_auc_m2: {roc_auc_m2}')
```

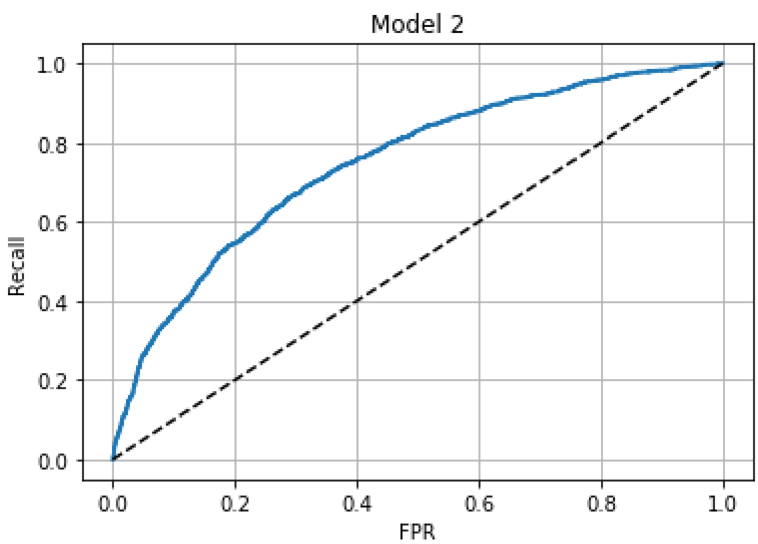
```
roc_auc_m1: 0.6739852539784691
roc_auc_m2: 0.7477612290491755
```

Ploting roc_auc

In [37]:

```python
from sklearn.metrics import roc_curve
fpr_m1, tpr_m1, thresholds_m1 = roc_curve(y_train, y_score_m1)
fpr_m2, tpr_m2, thresholds_m2 = roc_curve(y_train, y_score_m2)


def plot_roc_curve(fpr, tpr,model,label=None):
    plt.title(model)
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal
    plt.ylabel('Recall')
    plt.xlabel('FPR')
    plt.grid()
    plt.show()

plot_roc_curve(fpr_m1, tpr_m1,model="Model 1")
plot_roc_curve(fpr_m2, tpr_m2,model="Model 2")


```





According to this plot the best model is the model which is far away from the dotted line. So in this case both our model is good we can use any of the two based on the precison or recall what we want more.

Taking this into consideration Using Model 2 for Application Purpose because we want high precision at 37 % recall

In [38]:

```
import pickle
file_name = 'Model.picle'
pickle.dump(model2,open(file_name,'wb'))
```

In [40]:

```
model = pickle.load(open(file_name,'rb'))
#model.predict([a])
```

Testing Score

In [41]:

```
model1.score(x_test,y_test)
```

Out[41]:

```
0.4340659340659341
```

In [42]:

```
model2.score(x_test,y_test)
```

Out[42]:

```
0.6978021978021978
```

EOF

In [ ]:

```

```