SML_Assignment_3

Name: Ayush Patel

Collabration: Abhinav Nippani, Adwait Patil

In [1]:
```
import pandas as pd
import numpy as np
import itertools
import os
import sklearn
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import statsmodels.api as sm
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import StratifiedKFold
from sklearn.utils import resample
from sklearn.tree import plot_tree
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import GridSearchCV
from sklearn.utils import resample
from sklearn.tree import DecisionTreeClassifier
from scipy import stats
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
import random
import scipy
from sklearn.preprocessing import OneHotEncoder
```

In [2]:
```
path = os.getcwd()
```

Problem 1

In [3]:
```
np.random.seed(123)
x = np.random.normal(0, 1, (200))
y = x + 2 * x**2 - 2 * x**3 + np.random.normal(0, 1, (200))
```

Bias column to ensure the columns are named properly

In [4]:
```
for polynomial in range(1,11):
    polynomial_features = PolynomialFeatures(degree = polynomial, include_bias = True)
    x_polynomial = polynomial_features.fit_transform(x.reshape(-1,1))
```

Removing the bias column

In [5]:
```
x_polynomial = pd.DataFrame(x_polynomial).iloc[:,1:]
display(x_polynomial)
```

	1	2	3	4	5	6	7	8	9	10
0	-1.085631	1.178594	-1.279518	1.389083	-1.508031	1.637165	-1.777356e+00	1.929553e+00	-2.094781e+00	2.274159e+00
1	0.997345	0.994698	0.992057	0.989424	0.986798	0.984178	9.815655e-01	9.789598e-01	9.763611e-01	9.737693e-01
2	0.282978	0.080077	0.022660	0.006412	0.001815	0.000513	1.453028e-04	4.111758e-05	1.163539e-05	3.292565e-06
3	-1.506295	2.268924	-3.417668	5.148015	-7.754428	11.680454	-1.759421e+01	2.650206e+01	-3.991991e+01	6.013115e+01
4	-0.578600	0.334778	-0.193703	0.112076	-0.064847	0.037521	-2.170953e-02	1.256114e-02	-7.267877e-03	4.205195e-03
...	...	...	...	...	...	...	...	...	...	...
195	-3.231055	10.439716	-33.731298	108.987680	-352.145189	1137.800476	-3.676296e+03	1.187831e+04	-3.837949e+04	1.240062e+05
196	-0.269293	0.072519	-0.019529	0.005259	-0.001416	0.000381	-1.027025e-04	2.765711e-05	-7.447880e-06	2.005666e-06
197	-0.110851	0.012288	-0.001362	0.000151	-0.000017	0.000002	-2.056694e-07	2.279860e-08	-2.527241e-09	2.801465e-10
198	-0.341262	0.116460	-0.039743	0.013563	-0.004628	0.001580	-5.390301e-04	1.839503e-04	-6.277520e-05	2.142277e-05
199	-0.217946	0.047501	-0.010353	0.002256	-0.000492	0.000107	-2.335856e-05	5.090910e-06	-1.109545e-06	2.418211e-07

200 rows × 10 columns

a)

In [6]:
```
list_of_columns = list(x_polynomial.columns)

subset_selection = {"Features" : [], "AIC" : [], "BIC" : [], "Adjusted_R_Square" : []}

for A in range(len(list_of_columns) + 1):
    for subset in itertools.combinations(list_of_columns, A):

        if(list(subset)!=[]):

            x_polynomial_subset = x_polynomial[list(subset)]
            subset_selection["Features"].append(list(subset))

            regr = sm.OLS(y, sm.add_constant(x_polynomial_subset)).fit()

            aic = regr.aic
            subset_selection["AIC"].append(aic)
            bic = regr.bic
            subset_selection["BIC"].append(bic)
            adj_r_square = regr.rsquared_adj
            subset_selection["Adjusted_R_Square"].append(adj_r_square)
```

In [7]:
```
subset_selection = pd.DataFrame(subset_selection)
display(subset_selection)
```

	Features	AIC	BIC	Adjusted_R_Square
0	[1]	1356.768181	1363.364816	0.421163
1	[2]	1387.524884	1394.121519	0.324938
2	[3]	1065.501335	1072.097970	0.865079
3	[4]	1330.869591	1337.466225	0.491468
4	[5]	1063.848982	1070.445617	0.866189
...	...	...	...	...
1018	[1, 2, 3, 5, 6, 7, 8, 9, 10]	552.240533	585.223707	0.990030
1019	[1, 2, 4, 5, 6, 7, 8, 9, 10]	558.460353	591.443527	0.989715
1020	[1, 3, 4, 5, 6, 7, 8, 9, 10]	558.104425	591.087599	0.989733
1021	[2, 3, 4, 5, 6, 7, 8, 9, 10]	557.049808	590.032982	0.989787
1022	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	552.185498	588.466989	0.990079

1023 rows × 4 columns

Finding features with minimum AIC.

In [8]:
```
subset_selection[subset_selection["AIC"] == subset_selection["AIC"].min()]
```

Out[8]:

	Features	AIC	BIC	Adjusted_R_Square
55	[1, 2, 3]	542.275105	555.468374	0.990236

Finding features with minimum BIC.

In [9]:
```
subset_selection[subset_selection["BIC"] == subset_selection["BIC"].min()]
```

Out[9]:

	Features	AIC	BIC	Adjusted_R_Square
55	[1, 2, 3]	542.275105	555.468374	0.990236

Finding features with maximum Adjusted R square.

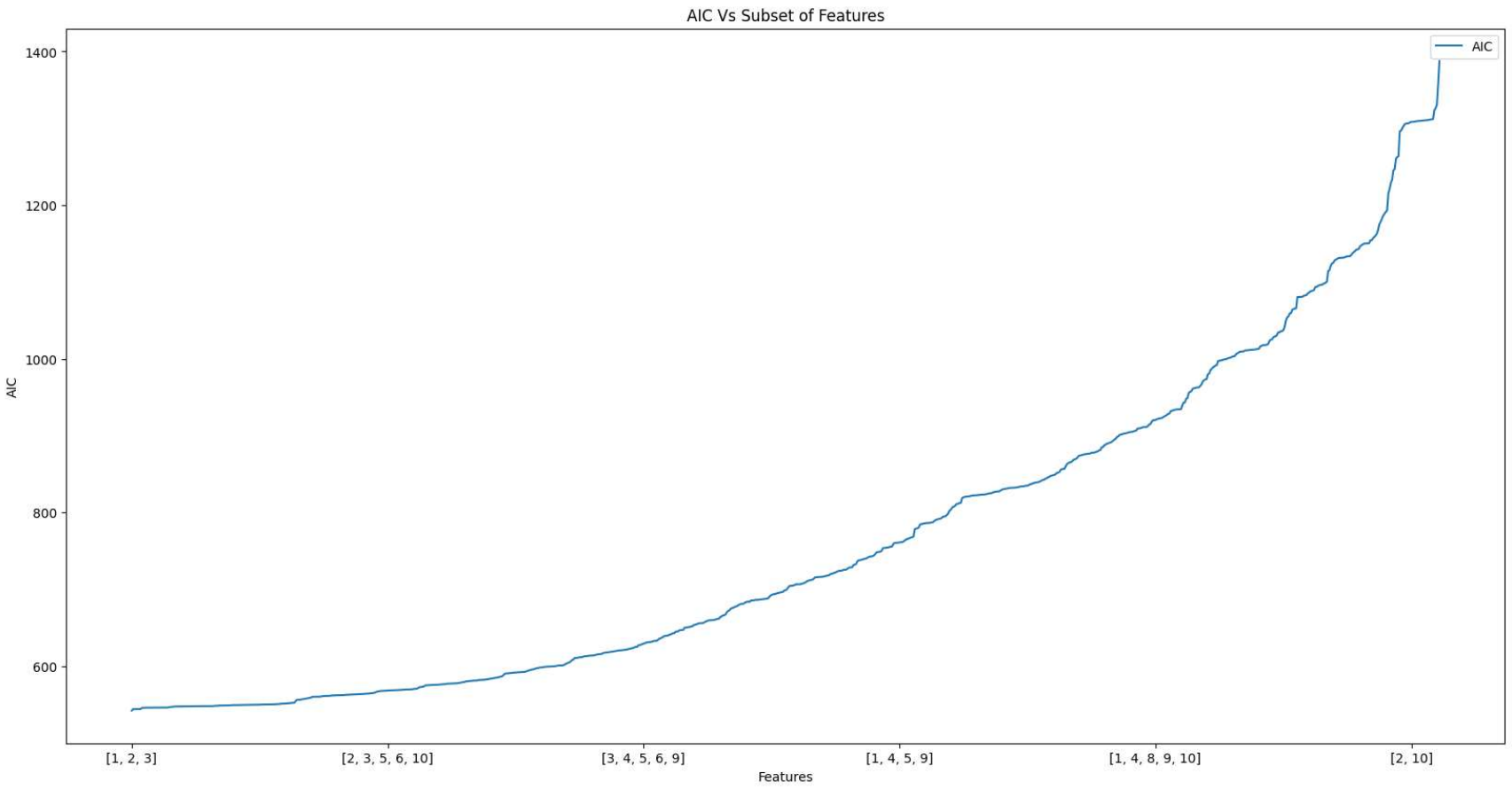
In [10]:
```
subset_selection[subset_selection["Adjusted_R_Square"] == subset_selection["Adjusted_R_Square"].max()]
```

Out[10]:

	Features	AIC	BIC	Adjusted_R_Square
55	[1, 2, 3]	542.275105	555.468374	0.990236

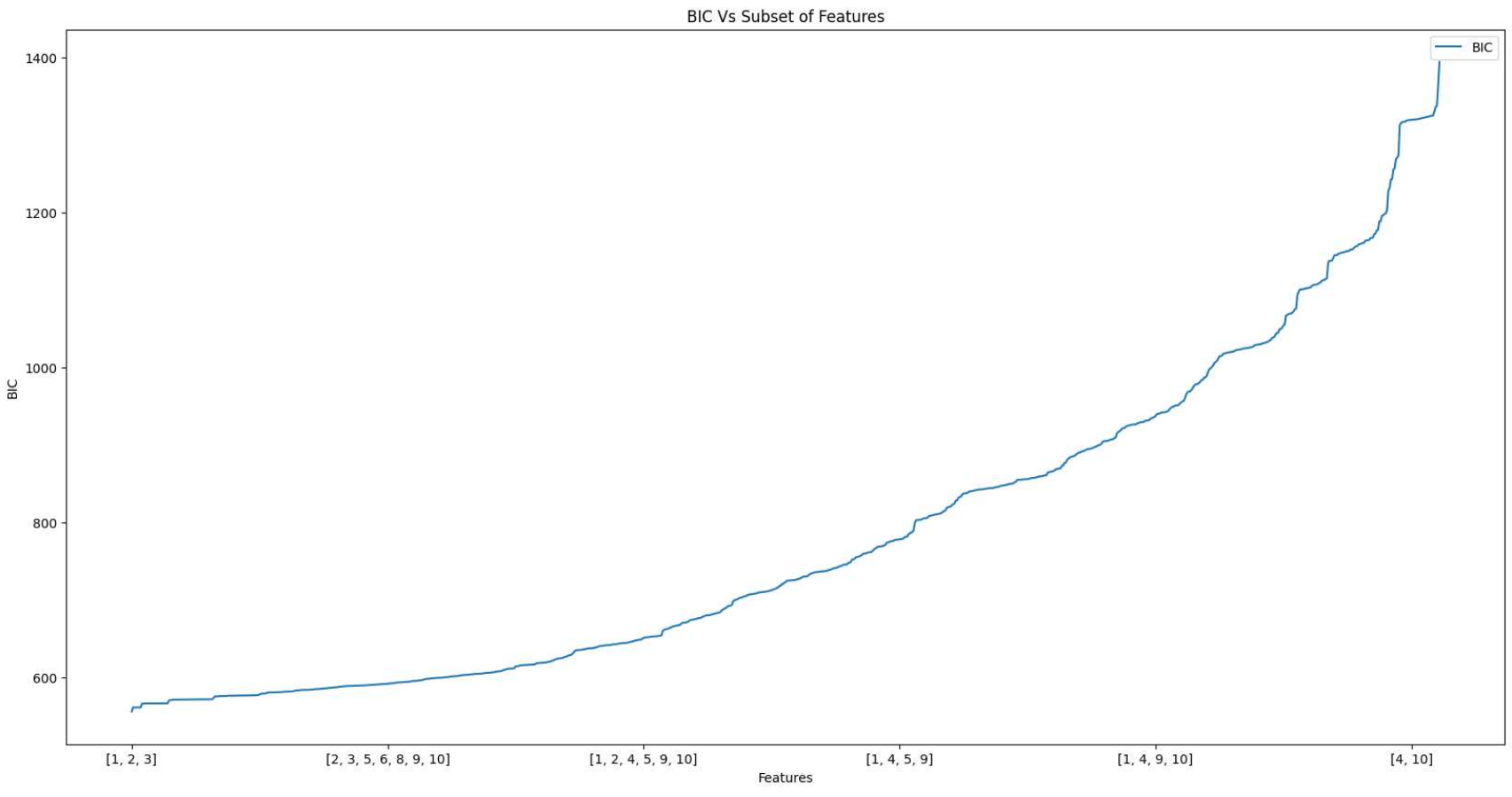
Plotting a graph of AIC vs Subset of Features

In [11]:
```
subset_selection.sort_values(["AIC"]).plot(x="Features",y="AIC",figsize=(20,10))
plt.xlabel("Features")
plt.ylabel("AIC")
plt.title("AIC Vs Subset of Features")
plt.show()
```



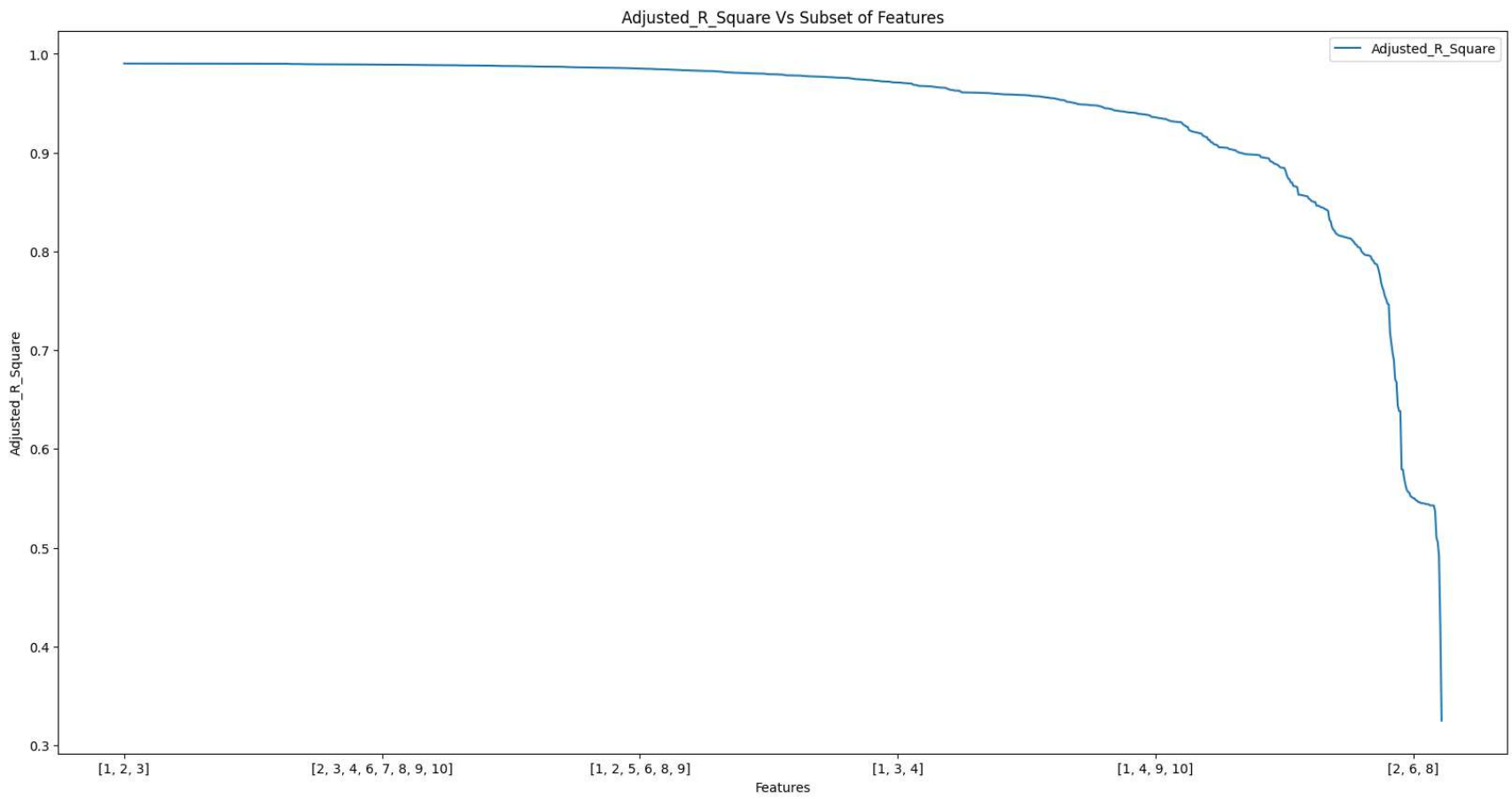
Plotting a graph of BIC vs Subset of Features

In [12]:
```
subset_selection.sort_values(["BIC"]).plot(x="Features",y="BIC",figsize=(20,10))
plt.xlabel("Features")
plt.ylabel("BIC")
plt.title("BIC Vs Subset of Features")
plt.show()
```



Plotting a graph of Adjusted_R_Square vs Subset of Features

In [13]:
```
subset_selection.sort_values(["Adjusted_R_Square"], ascending = False).plot(x="Features",y="Adjusted_R_Squa
plt.xlabel("Features")
plt.ylabel("Adjusted_R_Square")
plt.title("Adjusted_R_Square Vs Subset of Features")
plt.show()
```



Observation:
All the above graphs plotted indiacte that the optimal features are [1,2,3]

Filtering the optimal features, then training the model using OLS

In [14]:
```
best_features = [1,2,3]
best_x_polynomial = x_polynomial[best_features]
regr = sm.OLS(y, sm.add_constant(best_x_polynomial)).fit()
```

Coefficients of the best model

In [15]:
```
pd.DataFrame(regr.params).T
```

Out[15]:

	const	1	2	3
0	-0.181156	0.914572	2.06306	-1.970858

b)

In [16]:
```
list_of_columns = list(x_polynomial.columns)
final_column = []
best_adj_r_square = -np.inf
for i in range(len(list_of_columns)):
    for column in list_of_columns:

        if(column not in final_column):
            x_polynomial_subset = x_polynomial[final_column + [column]]
            regr = sm.OLS(y, sm.add_constant(x_polynomial_subset)).fit()
            adj_r_square = regr.rsquared_adj

            if(adj_r_square > best_adj_r_square):
                #print(adj_r_square,column)
                best_adj_r_square = adj_r_square
                best_column = column

    if(best_column not in final_column):
        final_column.append(best_column)
```

In [17]:
```
final_column
```

Out[17]: [5, 2, 3, 1]

Observation:
We can see that after every iteration, one feature that contributes the most is being added to the selection which can be seen with the increase in adjusted 12 values at every step. The selection stops after 4 iterations since there is no improvement in the model performance after 4 features i.e; [5,2,3,1]

Although, from subset selection, we know that it is not the most optimal model, there is not a significant difference in the performance between the models trained on the obtained subsets (best subset selection and forward subset selection). However, the computational time significantly reduces in the case of forward subset selection

c)

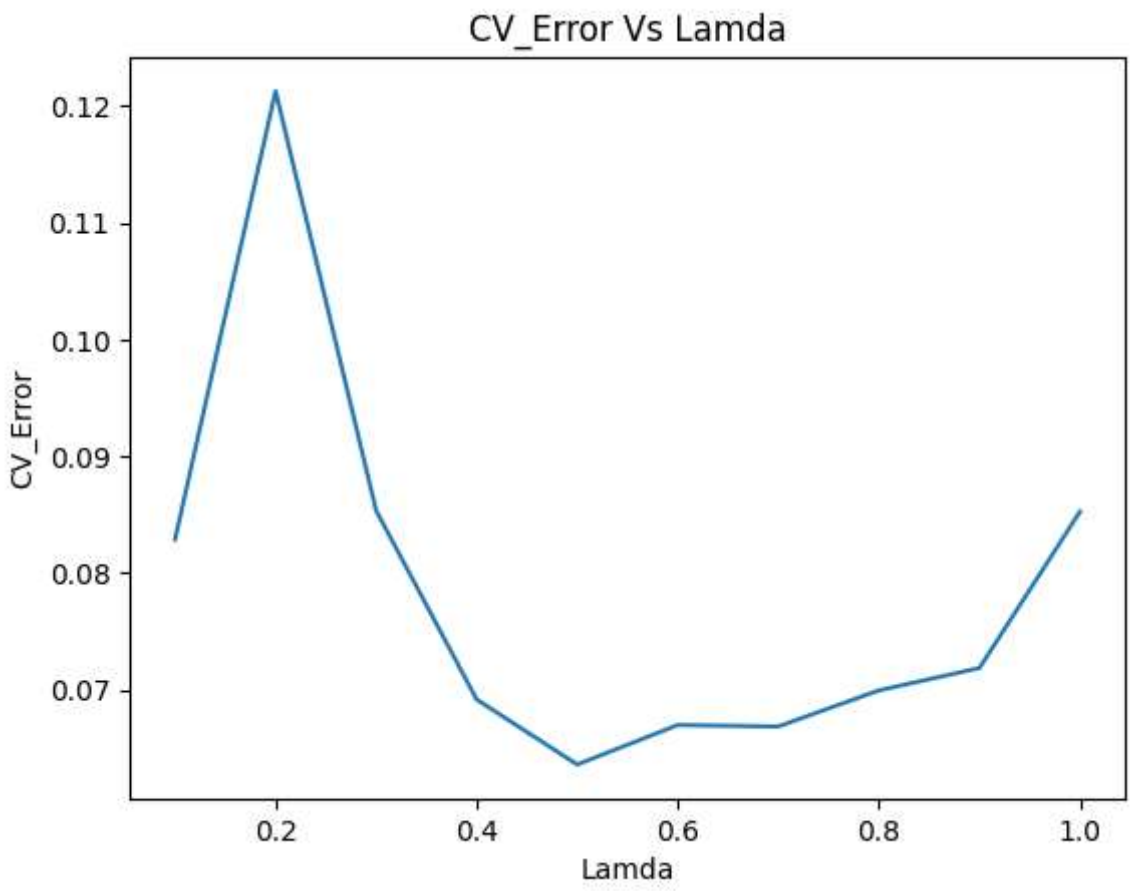
In [18]:
```
cv_error_data = {"Lamda":[], "CV_Error":[]}


for alpha in np.arange(0.1,1.1,0.1):
    #print(alpha)
    clf = linear_model.Lasso(alpha=alpha,max_iter = 100000000)

    best_cv_error = 1 - cross_val_score(clf, x_polynomial, y, scoring = "r2").mean()

    cv_error_data["Lamda"].append(alpha)
    cv_error_data["CV_Error"].append(best_cv_error)


cv_error_data = pd.DataFrame(cv_error_data)
```

In [19]:
```
cv_error_data[cv_error_data["CV_Error"] == cv_error_data["CV_Error"].min()]
```

Out[19]:

	Lamda	CV_Error
4	0.5	0.063629

In [20]:
```
plt.plot(cv_error_data["Lamda"],cv_error_data["CV_Error"])
plt.xlabel("Lamda")
plt.ylabel("CV_Error")
plt.title("CV_Error Vs Lamda")
plt.show()
```



Filtering the best features and printing their co-efficients

In [21]:
```
regr = linear_model.Lasso(alpha=0.5,max_iter = 1000000).fit(x_polynomial,y)
coefficient = pd.DataFrame(regr.coef_).T
coefficient.columns = x_polynomial.columns
coefficient
```

Out[21]:

	1	2	3	4	5	6	7	8	9	10
0	-0.0	0.0	-0.0	0.670943	-0.635998	0.0	0.048937	-0.021227	0.00071	0.001987

In [22]:
```
regr.intercept_
```

Out[22]:
```
0.5804372836802458
```

Problem 2

In [23]:
```
train = pd.read_csv('D:\SML\Train_Data.csv')
test = pd.read_csv('D:\SML\Test_Data.csv')
```

In [24]:
```
train
```

Out[24]:

	age	sex	bmi	smoker	region	children	charges
0	21.000000	male	25.745000	no	northeast	2	3279.868550
1	36.976978	female	25.744165	yes	southeast	3	21454.494239
2	18.000000	male	30.030000	no	southeast	1	1720.353700
3	37.000000	male	30.676891	no	northeast	3	6801.437542
4	58.000000	male	32.010000	no	southeast	1	11946.625900
...	...	...	...	...	...	...	...
3625	48.820767	female	41.426984	no	northwest	4	10987.324964
3626	38.661977	female	26.202557	no	southeast	2	11735.844352
3627	56.000000	male	40.300000	no	southwest	0	10602.385000
3628	48.061207	female	34.930624	no	southeast	1	8976.140452
3629	37.598865	female	25.219233	no	northeast	3	7027.698968

3630 rows × 7 columns

In [25]:
```
X = train.drop(columns=["charges"])
y = train[["charges"]]
```

Performing One Hot Encoding

In [26]:
```
def encoding(df,col_name):
    dummy = pd.get_dummies(df[col_name]).iloc[:,1:]
    dummy.columns = [col_name+"_" + str(x) for x in dummy.columns]

    df = df.drop(col_name,axis = 1)

    df = df.join(dummy)


    return df
```

In [27]:
```
X = encoding(X,"sex")
X = encoding(X,"smoker")
X = encoding(X,"region")
```

a)

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

b)

In [29]:
```
regr = sm.OLS(y_train, X_train).fit()
y_pred = regr.predict(X_test)
```

In [30]:
```
print("The mean squared error(test error) is: {}".format(mean_squared_error(y_pred, y_test, squared=False))
```

```
The mean squared error(test error) is: 5860.094769311387
```

c)

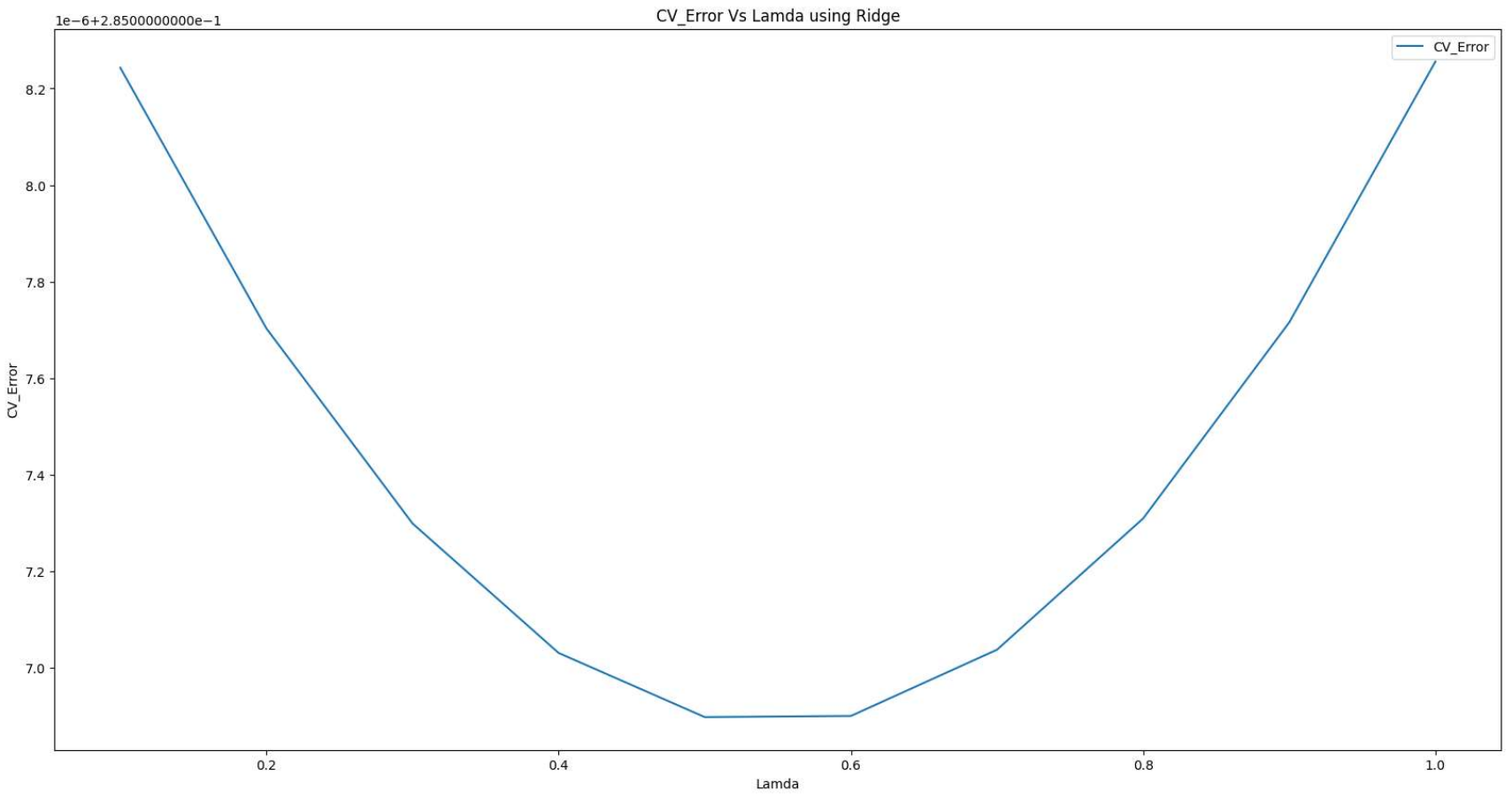
In [31]:
```
cross_val = {"Lamda":[], "CV_Error":[]}

for lamda in np.arange(0.1, 1.1, 0.1):

    regr = linear_model.Ridge(alpha = lamda)

    cv_error = 1 - cross_val_score(regr, X_train, y_train, scoring="r2").mean()

    cross_val["Lamda"].append(lamda)
    cross_val["CV_Error"].append(cv_error)

cross_val = pd.DataFrame(cross_val)
cross_val
```

Out[31]:

	Lamda	CV_Error
0	0.1	0.285008
1	0.2	0.285008
2	0.3	0.285007
3	0.4	0.285007
4	0.5	0.285007
5	0.6	0.285007
6	0.7	0.285007
7	0.8	0.285007
8	0.9	0.285008
9	1.0	0.285008

Plotting the graph to fing the minimal cv error

In [32]:
```
cross_val.plot(x="Lamda", y="CV_Error",figsize=(20,10))
plt.xlabel("Lamda")
plt.ylabel("CV_Error")
plt.title("CV_Error Vs Lamda using Ridge")
plt.show()
```



In [33]:
```
min_cv_error = cross_val[cross_val["CV_Error"] == cross_val["CV_Error"].min()]
min_cv_error
```

Out[33]:

	Lamda	CV_Error
4	0.5	0.285007

The optimal value of lamda obtained is 0.5

In [34]:
```
regr = linear_model.Ridge(alpha=min_cv_error["Lamda"].values[0]).fit(X_train,y_train)

y_pred = regr.predict(X_test)
```

In [35]:
```
print("The mean squared error(test error) for Ridge is: {}".format(mean_squared_error(y_pred, y_test, squar
```

```
The mean squared error(test error) for Ridge is: 5671.667330077637
```

d)

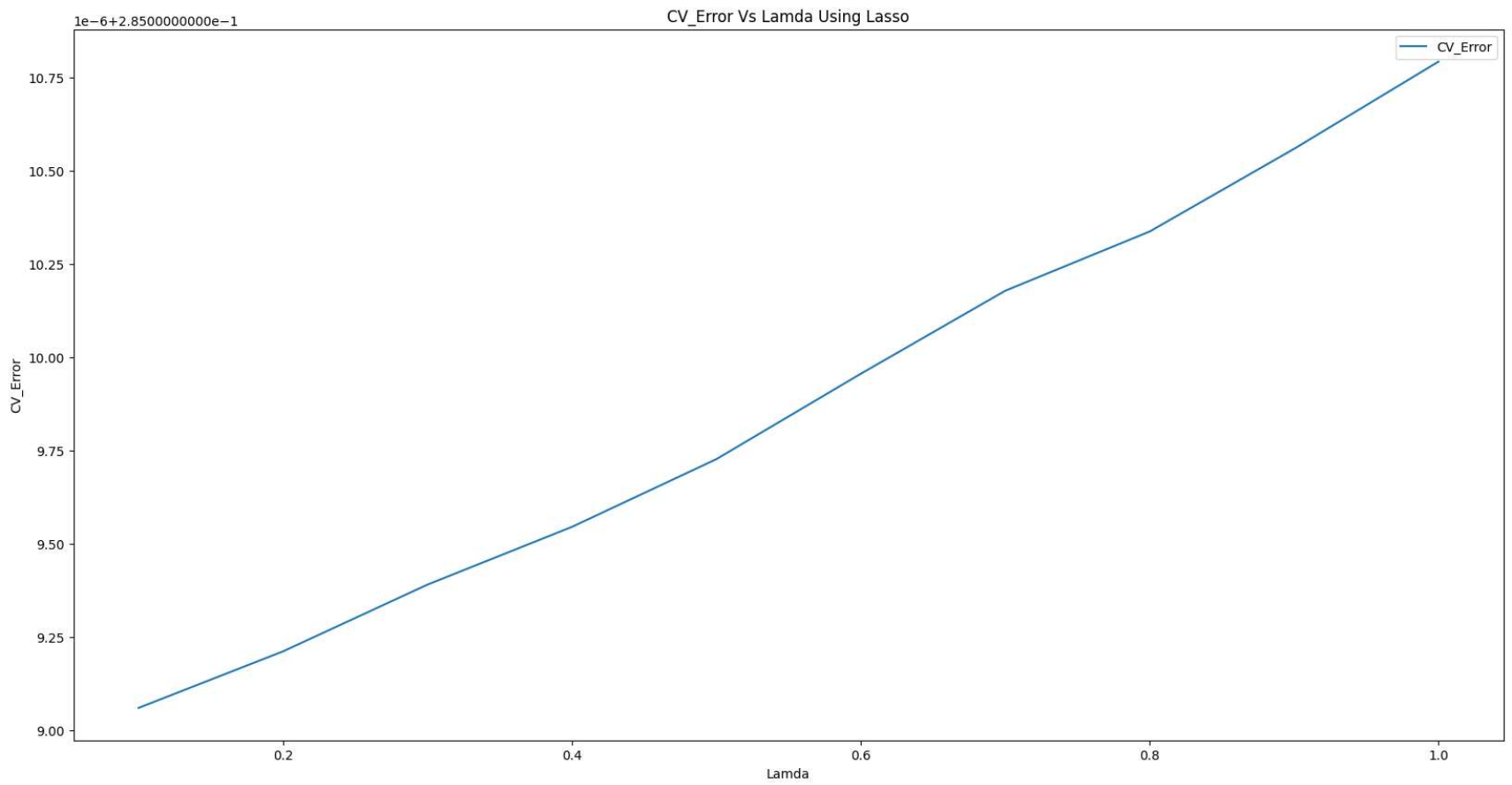
In [36]:
```
cross_val = {"Lamda":[], "CV_Error":[]}

for lamda in np.arange(0.1, 1.1, 0.1):
    regr = linear_model.Lasso(alpha = lamda)

    cv_error = 1 - cross_val_score(regr, X_train, y_train, scoring="r2").mean()

    cross_val["Lamda"].append(lamda)
    cross_val["CV_Error"].append(cv_error)

cross_val = pd.DataFrame(cross_val)
cross_val
```

Out[36]:

	Lamda	CV_Error
0	0.1	0.285009
1	0.2	0.285009
2	0.3	0.285009
3	0.4	0.285010
4	0.5	0.285010
5	0.6	0.285010
6	0.7	0.285010
7	0.8	0.285010
8	0.9	0.285011
9	1.0	0.285011

Plotting the graph to fing the minimal cv error

In [37]:
```
cross_val.plot(x="Lamda", y="CV_Error",figsize=(20,10))
plt.xlabel("Lamda")
plt.ylabel("CV_Error")
plt.title("CV_Error Vs Lamda Using Lasso")
plt.show()
```



In [38]:
```
min_cv_error = cross_val[cross_val["CV_Error"] == cross_val["CV_Error"].min()]
min_cv_error
```

Out[38]:

	Lamda	CV_Error
0	0.1	0.285009

The optimal value of lamda obtained is 0.1

In [39]:
```
regr = linear_model.Lasso(alpha=min_cv_error["Lamda"].values[0]).fit(X_train,y_train)

y_pred = regr.predict(X_test)
```

In [40]:
```
print("The mean squared error(test error) for Lasso is: {}".format(mean_squared_error(y_pred, y_test, squar
```

```
The mean squared error(test error) for Lasso is: 5670.879834633087
```

In [41]:
```
coefficient = pd.DataFrame(regr.coef_).T
coefficient.columns = X_train.columns
coefficient
```

Out[41]:

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
0	235.507643	292.260956	507.388153	960.624686	22623.46324	-2276.372029	-1908.611075	-2319.871851

There are 8 non-zero coefficient estimates

In [42]:
```
regr.intercept_
```

Out[42]:
```
array([-9038.37916747])
```

e)

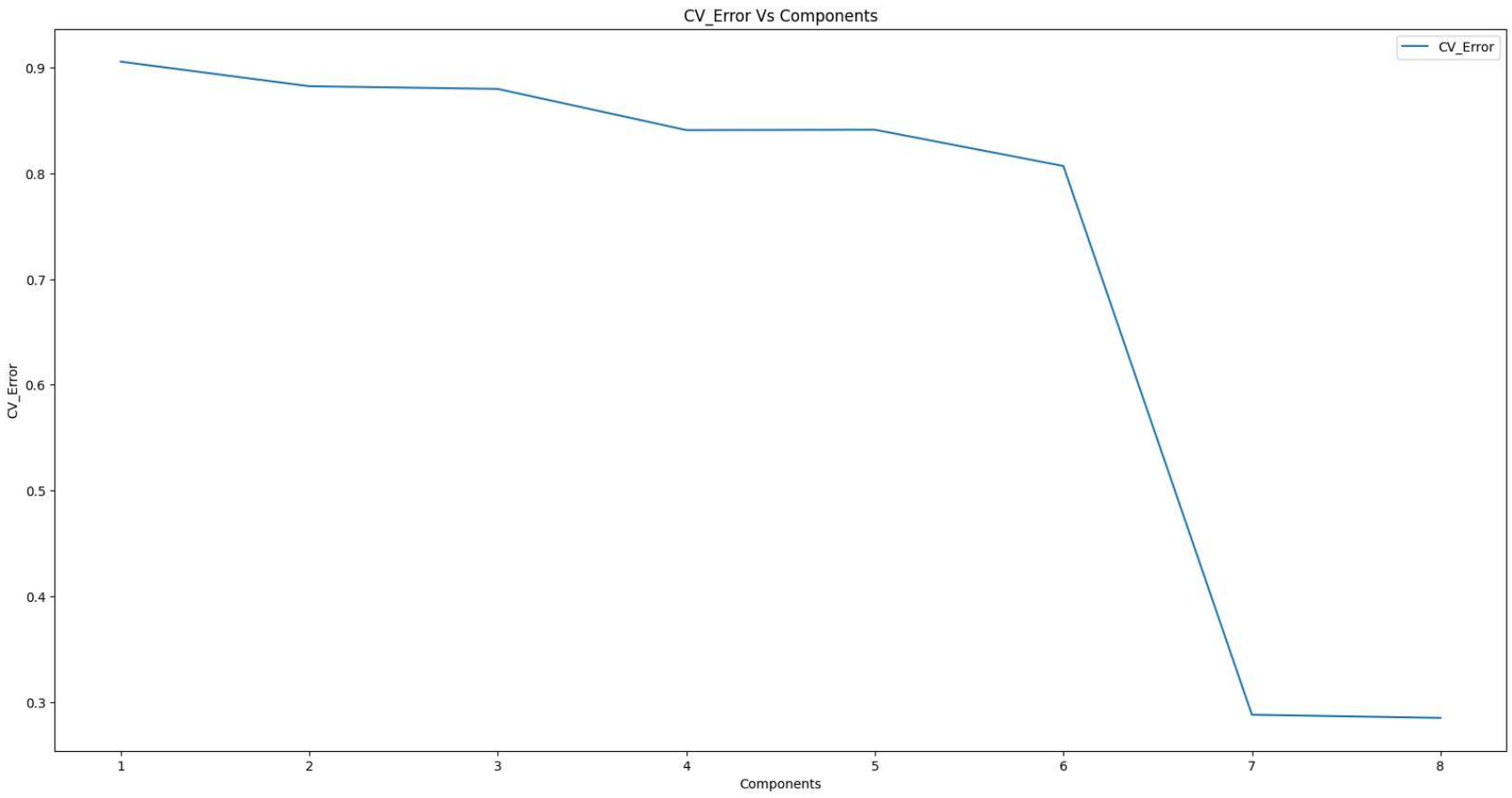
In [43]:
```
cross_val = {"Components":[], "CV_Error":[]}

for c in range(1,X_train.shape[1]+1):
    pca = PCA(n_components = c)
    X_train_reduced = pca.fit_transform(X_train)
    regr = linear_model.LinearRegression()
    cv_error = 1 - cross_val_score(regr,X_train_reduced,y_train, scoring = "r2").mean()

    cross_val["Components"].append(c)
    cross_val["CV_Error"].append(cv_error)

cross_val = pd.DataFrame(cross_val)
cross_val
```

Out[43]:

	Components	CV_Error
0	1	0.905684
1	2	0.882465
2	3	0.879893
3	4	0.840989
4	5	0.841323
5	6	0.806988
6	7	0.288013
7	8	0.285009

Plotting the graph to fing the minimal cv error

In [44]:
```
cross_val.plot(x="Components", y="CV_Error",figsize=(20,10))
plt.xlabel("Components")
plt.ylabel("CV_Error")
plt.title("CV_Error Vs Components")
plt.show()
```



By plotting a graph between the Number of Principal Components and the RMSE Score we can see that from 6 onwards there is a significant drop in RMSE and from 7 and 8 the value does not change much so we can safely say that it does not vary much when it is at 7 or 8.

In [45]:
```
min_cv_error = cross_val[cross_val["CV_Error"] == cross_val["CV_Error"].min()]
min_cv_error
```

Out[45]:

	Components	CV_Error
7	8	0.285009

The optimal component obtained is 8.

In [46]:
```
pca = PCA(n_components = min_cv_error["Components"].values[0])
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)
regr = linear_model.LinearRegression().fit(X_train_reduced,y_train)
y_pred = regr.predict(X_test_reduced)
```

In [47]:
```
print("The mean squared error(test error) for PCA is: {}".format(mean_squared_error(y_pred, y_test, squared
```

```
The mean squared error(test error) for PCA is: 5670.859881750769
```

f)

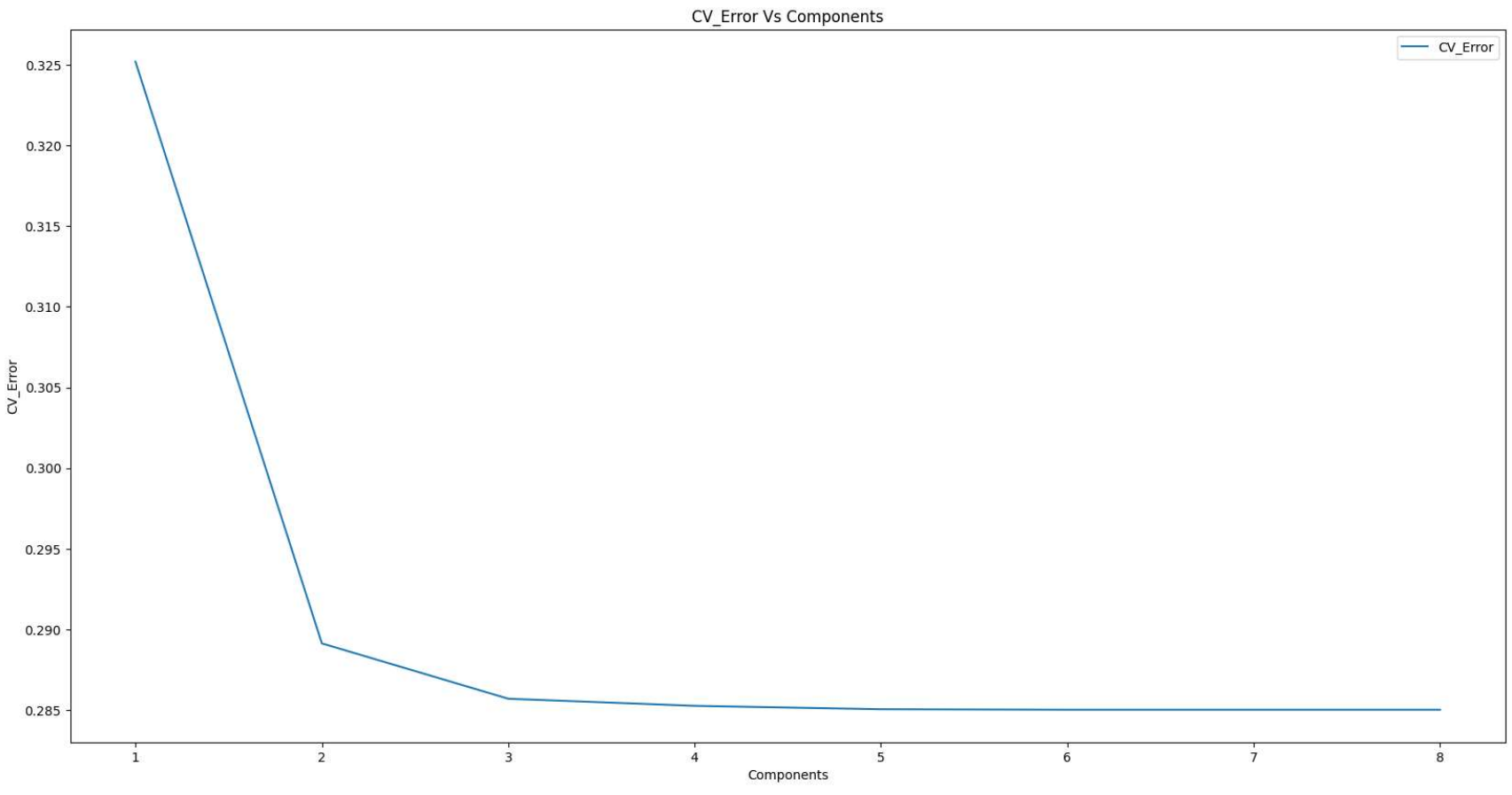
In [48]:
```
cross_val = {"Components":[], "CV_Error":[]}

# Iterating through no of components
for c in range(1,X_train.shape[1]+1):
    pls = PLSRegression(n_components = c)
    cv_error = 1 - cross_val_score(pls, X_train, y_train, scoring = "r2").mean()

    cross_val["Components"].append(c)
    cross_val["CV_Error"].append(cv_error)

cross_val = pd.DataFrame(cross_val)
cross_val
```

Out[48]:

	Components	CV_Error
0	1	0.325200
1	2	0.289125
2	3	0.285693
3	4	0.285250
4	5	0.285040
5	6	0.285010
6	7	0.285009
7	8	0.285009

Plotting the graph to fing the minimal cv error

In [49]:
```
cross_val.plot(x="Components", y="CV_Error",figsize=(20,10))
plt.xlabel("Components")
plt.ylabel("CV_Error")
plt.title("CV_Error Vs Components")
plt.show()
```



In [50]:
```
min_cv_error = cross_val[cross_val["CV_Error"] == cross_val["CV_Error"].min()]
min_cv_error
```

Out[50]:

	Components	CV_Error
7	8	0.285009

The optimal component obtained is 8.

In [51]:
```
pls = PLSRegression(n_components = min_cv_error["Components"].values[0])
regr.fit(X_train,y_train)

y_pred = regr.predict(X_test)
```

In [52]:
```
print("The mean squared error(test error) for PLSRegression is: {}".format(mean_squared_error(y_pred, y_tes
```

```
The mean squared error(test error) for PLSRegression is: 5670.859881750768
```

Problem 3

In [53]:
```
df = pd.read_csv("D:\SML\College.csv")
```

In [54]:
```
df['Accept.Rate'] = df['Accept'] / df['Apps']
```

In [55]:
```
df.head()
```

Out[55]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	I
0	Abilene Christian University	Yes	1660	1232	721	23	52	2885	537	7440	3300	450	2200	
1	Adelphi University	Yes	2186	1924	512	16	29	2683	1227	12280	6450	750	1500	
2	Adrian College	Yes	1428	1097	336	22	50	1036	99	11250	3750	400	1165	
3	Agnes Scott College	Yes	417	349	137	60	89	510	63	12960	5450	450	875	
4	Alaska Pacific University	Yes	193	146	55	16	44	249	869	7560	4120	800	1500	

In [56]:
```
df = pd.get_dummies(df, columns = ['Private'])
```

a)

In [57]:
```
X = df.drop(['Accept.Rate', 'Apps', 'Accept', 'Unnamed: 0', 'Private_No'], axis = 1)
y = df['Accept.Rate']
```

In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

In [59]:
```
X_train
```

Out[59]:

	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	Terminal	S.F.Ratio	perc.alu
739	478	12	25	2138	227	4470	2890	600	1210	33	33	16.3	
133	249	23	57	1698	894	9990	5666	800	1500	66	71	14.3	
234	198	7	20	545	42	11750	2700	400	850	77	83	14.0	
55	156	25	55	421	27	6500	2700	500	1000	76	76	14.3	
639	588	56	86	1846	154	9843	3180	600	1500	74	78	14.6	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
71	313	71	95	1088	16	18165	6750	500	1200	100	100	12.3	
106	288	55	82	943	7	11850	4270	600	900	95	99	11.4	
270	471	55	86	1818	23	14360	4090	400	650	77	92	12.9	
435	351	22	44	1419	228	6400	3150	500	1900	58	64	16.2	
102	902	6	24	6394	3881	5962	4444	500	985	69	73	16.7	

621 rows × 16 columns

b)

In [60]:
```
model = DecisionTreeRegressor()
model.fit(X_train, y_train)
```

Out[60]: DecisionTreeRegressor()

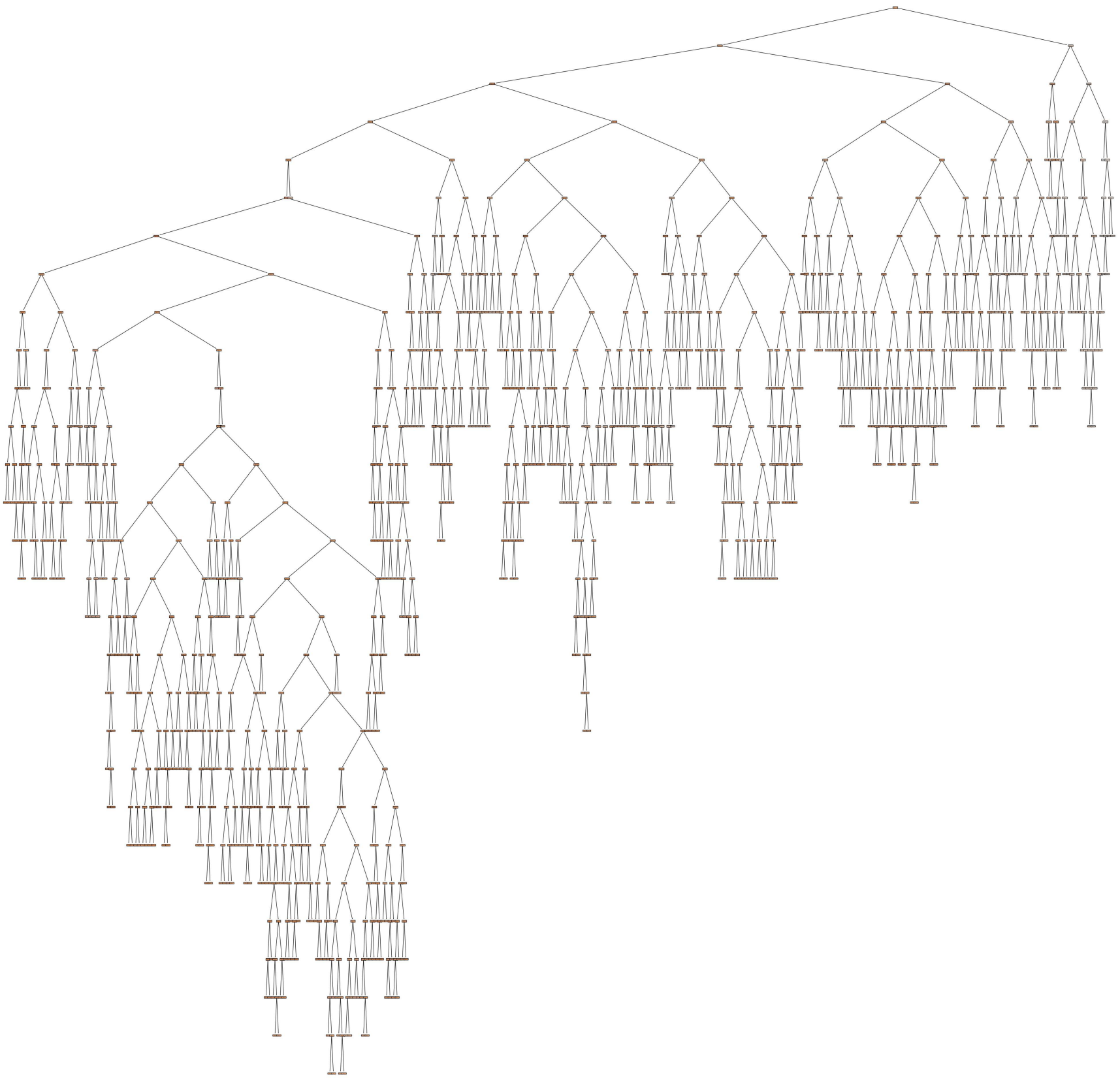
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:
```
y_predictions = model.predict(X_test)
```

In [62]:
```
print("The mean squared error for Decision Tree Regressor is: {}".format(mean_squared_error(y_predictions,
```

```
The mean squared error for Decision Tree Regressor is: 0.027824697997187
```

In [63]:
```
fig = plt.figure(figsize = (50, 50))
plot_tree(model, feature_names = X_train.columns, filled = True)
plt.show()
```



In [64]:
```
print("The mean squared error of the test set is: {}".format(mean_squared_error(y_test, y_predictions, squa
```

```
The mean squared error of the test set is: 0.027824697997187
```

Observation:
The tree plotted above has a large depth which makes it difficult for visualization.
Since, it is a big tree, there are a large number of parameters which might mean that the model is overfitting on the data. By reducing the depth, the performance of the model might improve further.
When we just pick a decision tree without specifying the depth, it would go till the maximum depth until values don't split further. As a result, we see that the decision tree is fully grown and it takes into account many features before coming to a conclusion for the acceptance rate.

The mean squared error of the test set is 0.0278246 respectively

c)

In [65]:
```
parameters = {'max_depth':range(1,30), 'min_samples_split':range(3,30), 'min_samples_leaf':range(3,30)}
clf = GridSearchCV(model, parameters, scoring='neg_mean_squared_error')
clf.fit(X_train, y_train)
```

Out[65]:
```
GridSearchCV(estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': range(1, 30),
                         'min_samples_leaf': range(3, 30),
                         'min_samples_split': range(3, 30)},
             scoring='neg_mean_squared_error')
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [66]:
```
clf.best_params_
```

Out[66]:
```
{'max_depth': 3, 'min_samples_leaf': 26, 'min_samples_split': 22}
```

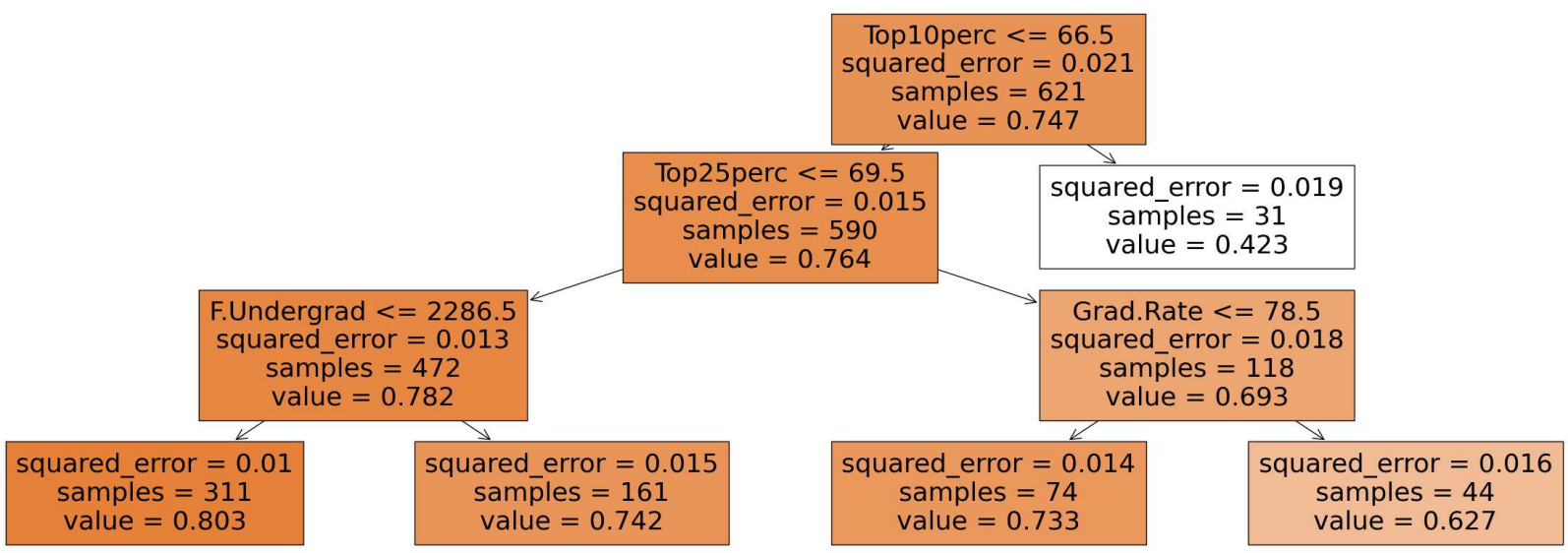
In [67]:
```
model = DecisionTreeRegressor(min_samples_split = clf.best_params_['min_samples_leaf'], min_samples_leaf =
```

In [68]:
```
model.fit(X_train, y_train)
y_predictions = model.predict(X_test)
```

In [69]:
```
print("The mean squared error of the best tree is: {}".format(mean_squared_error(y_predictions, y_test), sq
```

```
The mean squared error of the best tree is: 0.02005733000440076
```

In [70]:
```
fig = plt.figure(figsize = (30, 10))
plot_tree(model, feature_names = X_train.columns, filled = True)
plt.show()
```



Observation:

Based on the results from getting the best tree, we see that the mean square error reduced from 0.0278 to 0.020 with the best values of max_depth, min_samples_split and min_samples_leaf respectively.

The plot that we see above is quite readable and it points to the best features that were important to split the data into categories for better interpretability. The plot that we've seen in b) model takes a lot of parameters and it reaches the full depth of the tree.

d)

In [71]:
```
training_data = pd.concat([X_train, y_train], axis = 1, ignore_index = True)
testing_data = pd.concat([X_test, y_test], axis = 1, ignore_index = True)
```

In [72]:
```
training_data_len = len(training_data)
```

In [73]:
```
testing_data_len = len(testing_data)
```

In [74]:
```
bagging = BaggingRegressor(base_estimator = DecisionTreeRegressor(max_depth = clf.best_params_['max_depth']
                           min_samples_leaf = clf.best_params_['min_samples_split'],
                           min_samples_split = clf.best_params_['min_samples_leaf']),
                           n_estimators = 20)

```

In [75]:
```
bagging.fit(X_train,y_train.values.ravel())
y_predictions = bagging.predict(X_test)
```

In [76]:
```
print("The Mean Squared Error (MSE) of Bagged Decision Tree is: {}".format(mean_squared_error(y_predictions
```

```
The Mean Squared Error (MSE) of Bagged Decision Tree is: 0.01952223508092106
```

In [77]:
```
imp_df = pd.DataFrame(model.feature_importances_)
imp_df.columns = ["Feature Importances"]
imp_df["Features"] = list(X_train.columns)
imp_df = imp_df[["Features","Feature Importances"]]
imp_df = imp_df.sort_values(["Feature Importances"],ascending=False)
imp_df
```

Out[77]:

	Features	Feature Importances
1	Top10perc	0.702555
2	Top25perc	0.152808
3	F.Undergrad	0.081307
14	Grad.Rate	0.063330
0	Enroll	0.000000
4	P.Undergrad	0.000000
5	Outstate	0.000000
6	Room.Board	0.000000
7	Books	0.000000
8	Personal	0.000000
9	PhD	0.000000
10	Terminal	0.000000
11	S.F.Ratio	0.000000
12	perc.alumni	0.000000
13	Expend	0.000000
15	Private_Yes	0.000000

Observation:

We see that the features Top25perc, Top10perc and F.Undergrad are important for our decision trees as can be seen above. The mean squared error obtained as a result of bagging is about 0.019 respectively.

e)

In [78]:
```
metrics_df = {}
metrics_df["Estimators"] = []
metrics_df["Max Features"] = []
metrics_df["Error"] = []
for estimators in [1,20,50,100,150,200,300]:
    for max_feat in range(1,X_train.shape[1] + 1):
        rfr = RandomForestRegressor(n_estimators = estimators, max_features = max_feat)
        rfr.fit(X_train,y_train.values.ravel())
        y_prediction = rfr.predict(X_test)
        error = mean_squared_error(y_test, y_prediction, squared = True)
        metrics_df["Estimators"].append(estimators)
        metrics_df["Max Features"].append(max_feat)
        metrics_df["Error"].append(error)
metrics_df = pd.DataFrame(metrics_df)
metrics_df
```

Out[78]:

	Estimators	Max Features	Error
0	1	1	0.031652
1	1	2	0.032917
2	1	3	0.030255
3	1	4	0.028350
4	1	5	0.030153
...	...	...	...
107	300	12	0.016030
108	300	13	0.016236
109	300	14	0.016052
110	300	15	0.015960
111	300	16	0.016168

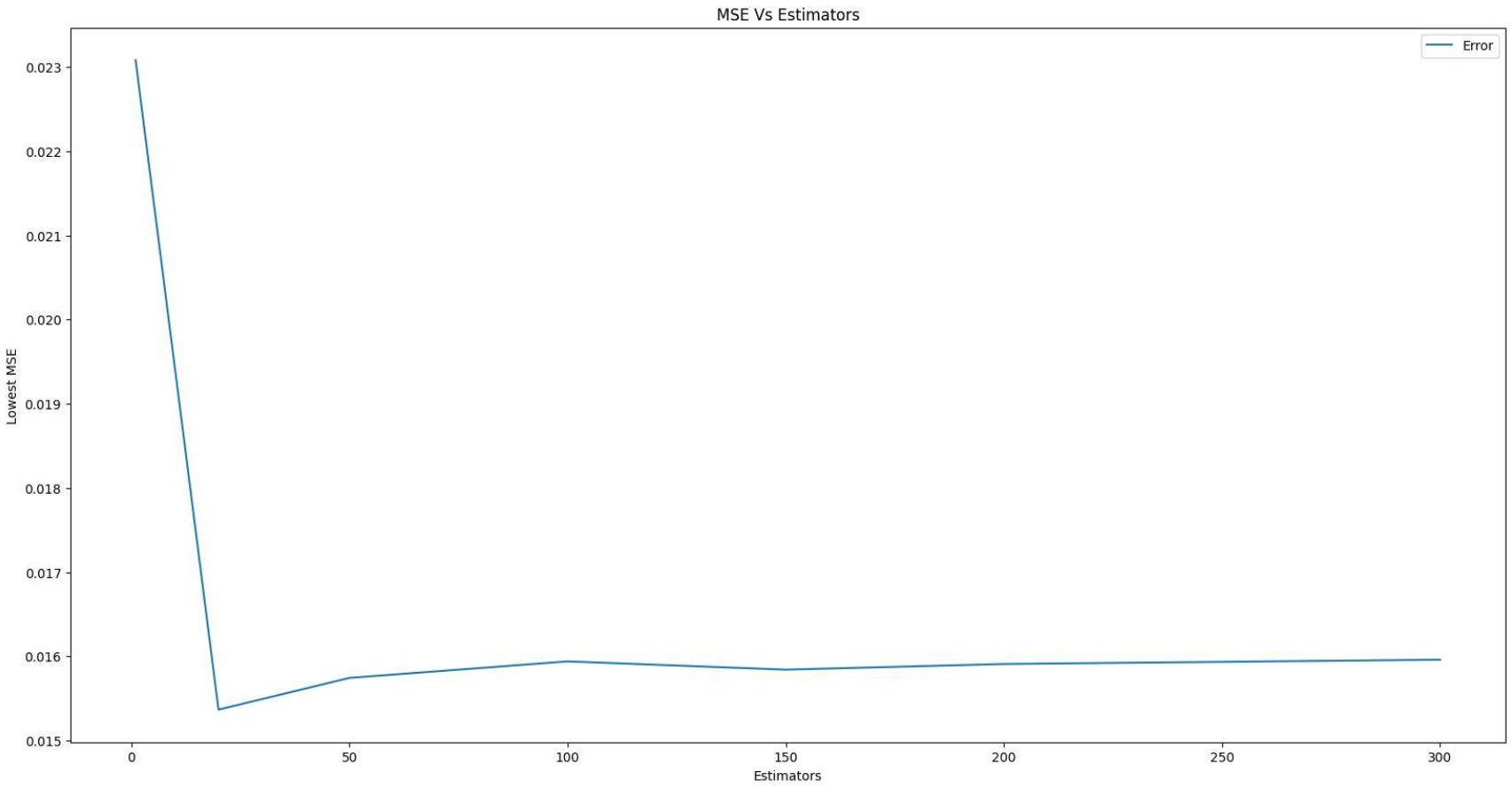
112 rows × 3 columns

In [79]:
```
best_params_df = metrics_df[metrics_df["Error"] == metrics_df["Error"].min()]
best_params_df
```

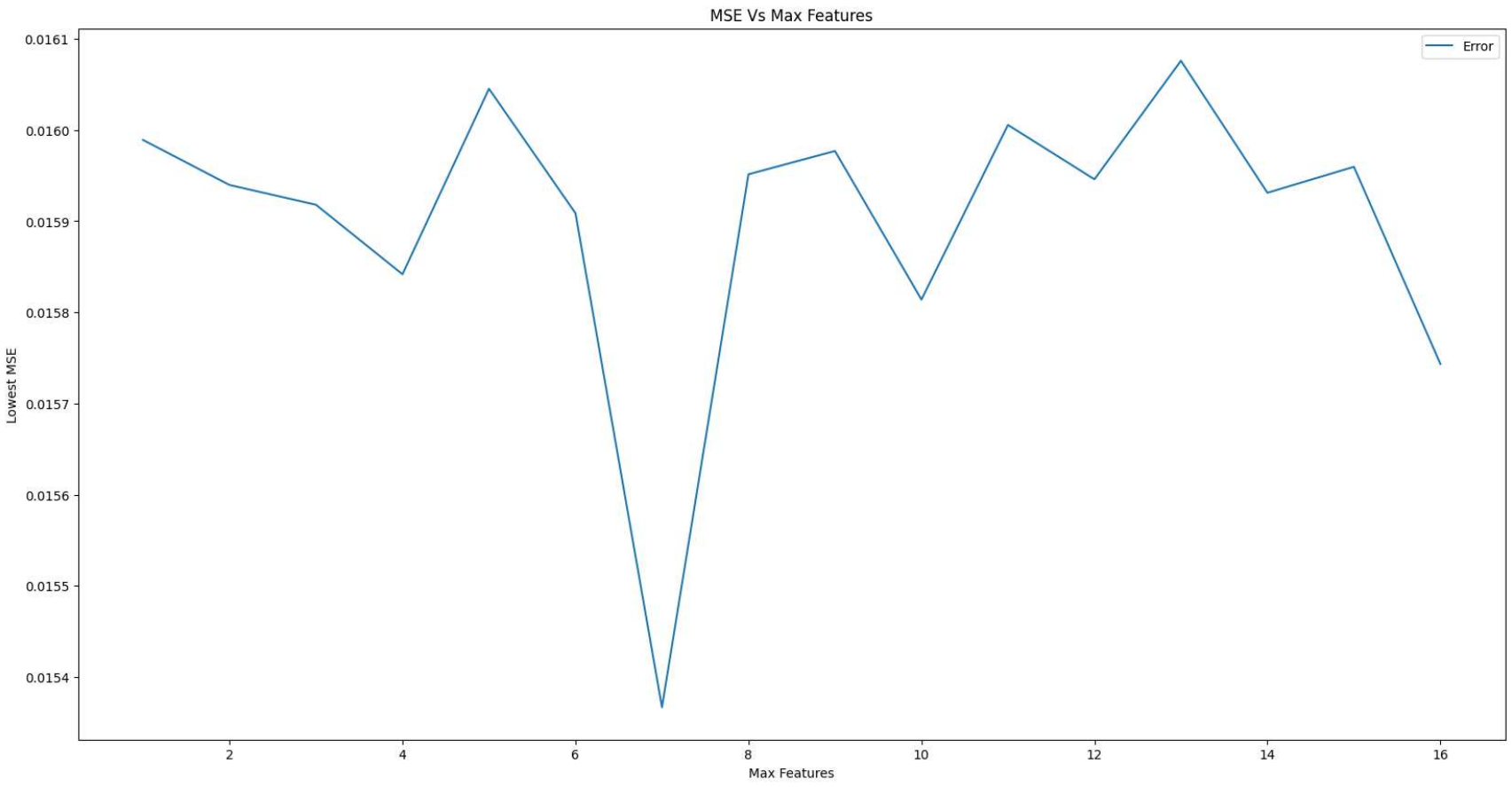
Out[79]:

	Estimators	Max Features	Error
22	20	7	0.015367

In [80]:
```
estimators_df = metrics_df.groupby(["Estimators"],as_index=False)["Error"].min()
estimators_df.plot(x="Estimators",y="Error",kind="line",figsize=(20,10))
plt.xlabel("Estimators")
plt.ylabel("Lowest MSE")
plt.title("MSE Vs Estimators")
plt.show()
```



In [81]:
```
 max_features_df = metrics_df.groupby(["Max Features"],as_index=False)["Error"].min()
max_features_df.plot(x="Max Features",y="Error",kind="line",figsize=(20,10))
plt.xlabel("Max Features")
plt.ylabel("Lowest MSE")
plt.title("MSE Vs Max Features")
plt.show()

```



In [82]:
```
rfr = RandomForestRegressor(n_estimators=best_params_df["Estimators"].values[0],
                            max_features = best_params_df["Max Features"].values[0])
rfr.fit(X_train, y_train.values.ravel())
```

Out[82]:
```
RandomForestRegressor(max_features=7, n_estimators=20)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [83]:
```
imp_df = pd.DataFrame(rfr.feature_importances_)
imp_df.columns = ["Feature Importances"]
imp_df["Features"] = list(X_train.columns)
imp_df = imp_df[["Features","Feature Importances"]]
imp_df = imp_df.sort_values(["Feature Importances"],ascending=False)
imp_df
```

Out[83]:

	Features	Feature Importances
2	Top25perc	0.159545
14	Grad.Rate	0.125136
1	Top10perc	0.124239
5	Outstate	0.081853
13	Expend	0.077530
3	F.Undergrad	0.065681
11	S.F.Ratio	0.054187
6	Room.Board	0.050510
7	Books	0.048101
4	P.Undergrad	0.041976
8	Personal	0.035931
12	perc.alumni	0.034739
0	Enroll	0.034697
10	Terminal	0.030795
9	PhD	0.029050
15	Private_Yes	0.006028

In [ ]:
```

```

In [84]:
```
print("The Mean Squared Error of the best Random Forest Regressor is: {}".format(mean_squared_error(y_predi
```

```
The Mean Squared Error of the best Random Forest Regressor is: 0.01952223508092106
```

Observation:

The most important features according to our random forest regressor are Top25perc, Top10perc and Grad.Rate respectively.

Problem 4

In [85]:
```
penguin_size = pd.read_csv('D:\SML\penguins_size.csv')
penguin_lter = pd.read_csv('D:\SML\penguins_lter.csv')
```

In [86]:
```
df = pd.concat([penguin_lter, penguin_size],axis=1)
df.head()
```

Out[86]:

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)	...	Delta 15 N (o/oo)	Delta 13 C (o/oo)	Comment
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1	...	NaN	NaN	Not enouc blood f isotope
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5	...	8.94956	-24.69454	Na
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3	...	8.36821	-25.33302	Na
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN	...	NaN	NaN	Adult n sample
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7	...	8.76651	-25.32426	Na

5 rows × 24 columns

In [87]:
```
df.columns
```

Out[87]:
```
Index(['studyName', 'Sample Number', 'Species', 'Region', 'Island', 'Stage',
       'Individual ID', 'Clutch Completion', 'Date Egg', 'Culmen Length (mm)',
       'Culmen Depth (mm)', 'Flipper Length (mm)', 'Body Mass (g)', 'Sex',
       'Delta 15 N (o/oo)', 'Delta 13 C (o/oo)', 'Comments', 'species',
       'island', 'culmen_length_mm', 'culmen_depth_mm', 'flipper_length_mm',
       'body_mass_g', 'sex'],
      dtype='object')
```

In [88]:
```
# Removing unnecessary columns
df = df.drop(columns=["Sample Number", "Individual ID", "Date Egg","Stage","Region","Comments"])

# Removing duplicate columns
df = df.drop(columns=["Island","Culmen Length (mm)", "Culmen Depth (mm)", "Flipper Length (mm)", "Body Mass

df
```

Out[88]:

	studyName	Clutch Completion	Delta 15 N (o/oo)	Delta 13 C (o/oo)	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_ma
0	PAL0708	Yes	NaN	NaN	Adelie	Torgersen	39.1	18.7	181.0	3
1	PAL0708	Yes	8.94956	-24.69454	Adelie	Torgersen	39.5	17.4	186.0	3
2	PAL0708	Yes	8.36821	-25.33302	Adelie	Torgersen	40.3	18.0	195.0	3
3	PAL0708	Yes	NaN	NaN	Adelie	Torgersen	NaN	NaN	NaN	
4	PAL0708	Yes	8.76651	-25.32426	Adelie	Torgersen	36.7	19.3	193.0	3
...	...	...	...	...	...	...	...	...	...	
339	PAL0910	No	NaN	NaN	Gentoo	Biscoe	NaN	NaN	NaN	
340	PAL0910	Yes	8.41151	-26.13832	Gentoo	Biscoe	46.8	14.3	215.0	4
341	PAL0910	Yes	8.30166	-26.04117	Gentoo	Biscoe	50.4	15.7	222.0	5
342	PAL0910	Yes	8.24246	-26.11969	Gentoo	Biscoe	45.2	14.8	212.0	5
343	PAL0910	Yes	8.36390	-26.15531	Gentoo	Biscoe	49.9	16.1	213.0	5

344 rows × 11 columns

In [89]:
```
df = df[~df["Delta 15 N (o/oo)"].isnull()].reset_index(drop = True)
df = df[~df["Delta 13 C (o/oo)"].isnull()].reset_index(drop = True)
df = df[~df["sex"].isnull()].reset_index(drop = True)
```

In [90]:
```
df["sex"].unique()
```

Out[90]:
```
array(['FEMALE', 'MALE', '.'], dtype=object)
```

In [91]:
```
df = df[~(df["sex"] == ".")].reset_index(drop = True)
```

In [92]:
```
df = encoding(df,"island")
df = encoding(df,"sex")
df = encoding(df,"Clutch Completion")
df = encoding(df,"studyName")
```

In [93]:
```
df["species"].unique()
```

Out[93]:
```
array(['Adelie', 'Chinstrap', 'Gentoo'], dtype=object)
```

In [94]:
```
df["species"] = df["species"].replace("Adelie",1)
df["species"] = df["species"].replace("Chinstrap",2)
df["species"] = df["species"].replace("Gentoo",3)
```

In [95]:
```
X = df.drop(columns=["species"])
Y = df[["species"]]
```

In [96]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

a)

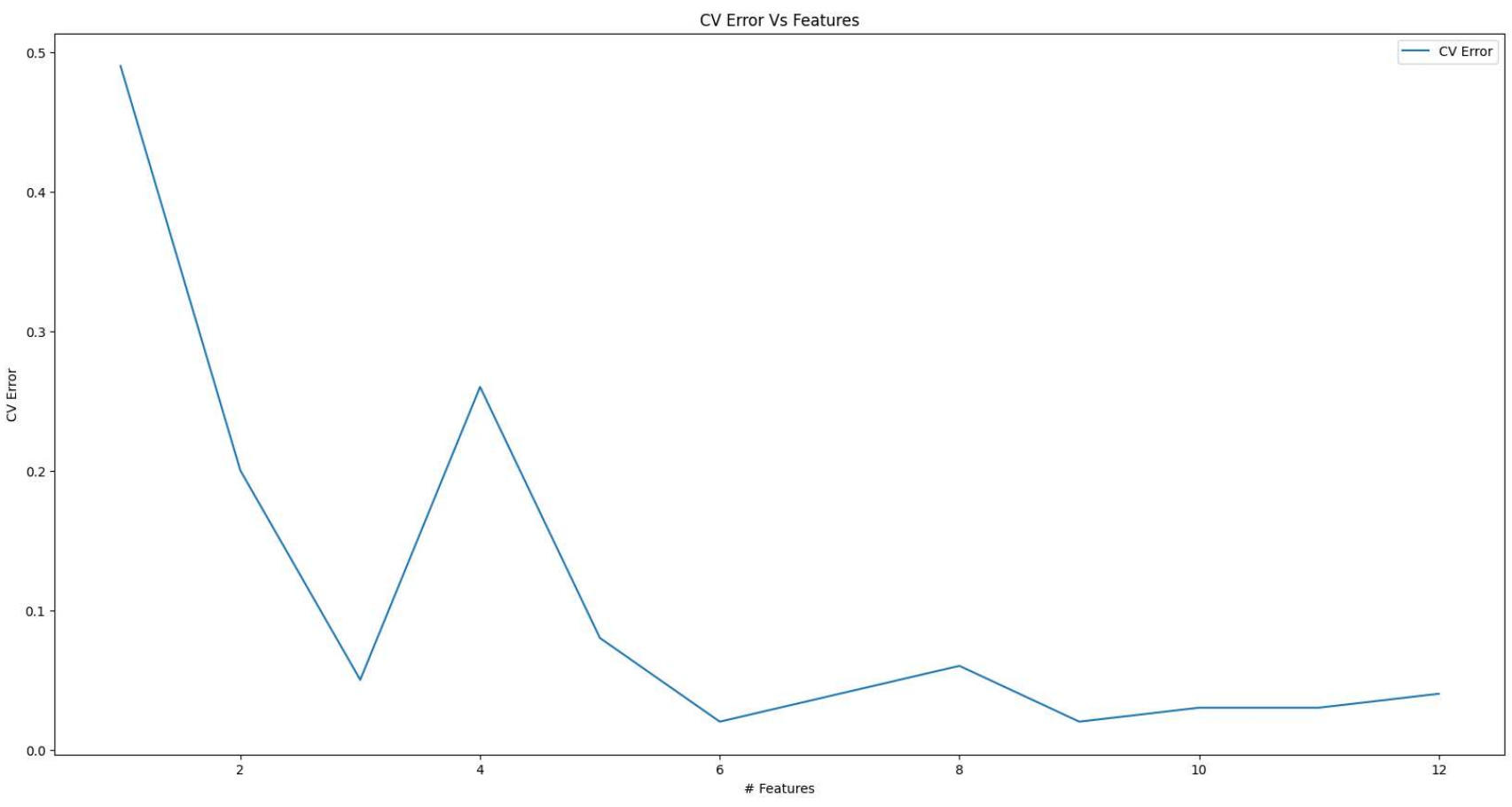
In [97]:
```
X_train_bootstrap, y_train_bootstrap = resample(X_train, y_train, n_samples = 100)
```

In [98]:
```
column_list = list(X_train_bootstrap.columns)

cv_df = {}
cv_df["Features"] = []
cv_df["# Features"] = []
cv_df["CV Error"] = []

# Iterating through all the combinations
for p in range(len(column_list) + 1):
    for subset in random.sample(list(itertools.combinations(column_list, p)),1):
        # Removing the empty subset
        if(list(subset)!=[]):

            # Filtering the subsetted dataframe
            X_subset = X_train_bootstrap[list(subset)]

            # Training a Decision Tree Classifier with max depth 6
            clf = DecisionTreeClassifier(max_depth = 6).fit(X_subset,y_train_bootstrap)

            # Finding the CV Error
            cv_error = 1 - cross_val_score(clf, X_subset, y_train_bootstrap, scoring = "accuracy").mean()

            cv_df["Features"].append(list(subset))
            cv_df["# Features"].append(len(list(subset)))
            cv_df["CV Error"].append(cv_error)

cv_df = pd.DataFrame(cv_df)
cv_df
```

Out[98]:

	Features	# Features	CV Error
0	[studyName_PAL0910]	1	0.49
1	[culmen_length_mm, island_Torgersen]	2	0.20
2	[Delta 15 N (o/oo), Delta 13 C (o/oo), body_ma...	3	0.05
3	[culmen_depth_mm, body_mass_g, sex_MALE, study...	4	0.26
4	[Delta 13 C (o/oo), culmen_depth_mm, island_To...	5	0.08
5	[Delta 15 N (o/oo), Delta 13 C (o/oo), flipper...	6	0.02
6	[culmen_length_mm, flipper_length_mm, body_mas...	7	0.04
7	[Delta 15 N (o/oo), Delta 13 C (o/oo), culmen_...	8	0.06
8	[Delta 15 N (o/oo), Delta 13 C (o/oo), culmen_...	9	0.02
9	[culmen_length_mm, culmen_depth_mm, flipper_le...	10	0.03
10	[Delta 15 N (o/oo), Delta 13 C (o/oo), culmen_...	11	0.03
11	[Delta 15 N (o/oo), Delta 13 C (o/oo), culmen_...	12	0.04

In [99]:
```
cv_df.plot(x="# Features",y="CV Error",figsize=(20,10))
plt.xlabel("# Features")
plt.ylabel("CV Error")
plt.title("CV Error Vs Features")
plt.show()
```



In [100]:
```
min_cv_df = cv_df[cv_df["CV Error"] == cv_df["CV Error"].min()].reset_index(drop=True)
display(min_cv_df)
print("\nTherefore, the most optimal p obtained is:",min_cv_df["# Features"].values[0])

```

	Features	# Features	CV Error
0	[Delta 15 N (o/oo), Delta 13 C (o/oo), flipper...	6	0.02
1	[Delta 15 N (o/oo), Delta 13 C (o/oo), culmen_...	9	0.02

```
Therefore, the most optimal p obtained is: 6
```

b)

In [101]:
```
y_pred_final_train = {}
y_pred_final_test = {}

# Calculating predictions for each tree
for n_trees in [1,50,100,150,200,300,400]:
    final_y_pred_train = []
    final_y_pred_test = []

    for i in range(n_trees):

        # Iterating through all the combinations of the best feature subset (p) obtained above
        for subset in random.sample(list(itertools.combinations(column_list, min_cv_df["# Features"].values

            # Filtering the subsetted dataframes
            X_train_bootstrap_subset = X_train_bootstrap[list(subset)]
            X_train_subset = X_train[list(subset)]
            X_test_subset = X_test[list(subset)]

            clf = DecisionTreeClassifier(max_depth = 6)
            clf.fit(X_train_bootstrap_subset, y_train_bootstrap)

            y_pred_train = clf.predict(X_train_subset)
            y_pred_test = clf.predict(X_test_subset)

            final_y_pred_train.append(y_pred_train)
            final_y_pred_test.append(y_pred_test)

    # Finding the mode of the predictions obtained
    y_pred_final_train[n_trees] = stats.mode(np.array(final_y_pred_train), keepdims = True)[0][0]
    y_pred_final_test[n_trees] = stats.mode(np.array(final_y_pred_test), keepdims = True)[0][0]

y_pred_final_train = pd.DataFrame(y_pred_final_train)
y_pred_final_test = pd.DataFrame(y_pred_final_test)

print("Reporting the final predictions of the Training Set for each tree in [1,50,100,150,200,300,400]:")
y_pred_final_train
```

```
Reporting the final predictions of the Training Set for each tree in [1,50,100,150,200,300,400]:
```

Out[101]:

	1	50	100	150	200	300	400
0	1	1	1	1	1	1	1
1	2	2	2	2	2	2	2
2	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1
4	2	2	2	2	2	2	2
...	...	...	...	...	...	...	...
254	2	2	2	2	2	2	2
255	1	1	1	1	1	1	1
256	1	1	1	1	1	1	1
257	3	3	3	3	3	3	3
258	1	1	1	1	1	1	1

259 rows × 7 columns

c)

In [102]:
```
metrics = {}
metrics["No of Trees"] = []
metrics["Train Zero One Test Error"] = []
metrics["Test Zero One Test Error"] = []
metrics["F1 Score"] = []
metrics["AUC"] = []

# For every T
for col in y_pred_final_train:

    metrics["No of Trees"].append(col)

    # Calculating the Train Error
    zero_one_error_train = 1 - (np.array(y_train["species"]) == np.array(y_pred_final_train[col])).sum()/le
    metrics["Train Zero One Test Error"].append(zero_one_error_train)

    # Calculating the Test Error
    zero_one_error_test = 1 - (np.array(y_test["species"]) == np.array(y_pred_final_test[col])).sum()/len(y_
    metrics["Test Zero One Test Error"].append(zero_one_error_test)

    # Calculating the F1 Score (One Vs All - 3 values for each class)
    f1score = f1_score(y_test["species"],y_pred_final_test[col],average = None)
    f1score = [round(x,3) for x in f1score]
    metrics["F1 Score"].append(f1score)

    # Calculating the AUC (One Vs All - 3 values for each class)
    y_pred_class = pd.DataFrame(OneHotEncoder().fit_transform(y_pred_final_test[[col]]).toarray()).astype(i
    y_test_class = pd.DataFrame(OneHotEncoder().fit_transform(y_test[["species"]]).toarray())

    auc_list = []
    for col1 in y_test_class.columns:
        auc = round(roc_auc_score(list(y_test_class[col1]),list(y_pred_class[col1])),3)
        auc_list.append(auc)

    metrics["AUC"].append(auc_list)

print("Reporting the Final Metrics for every tree(T): ")
pd.DataFrame(metrics)
```

```
Reporting the Final Metrics for every tree(T):
```

Out[102]:

	No of Trees	Train Zero One Test Error	Test Zero One Test Error	F1 Score	AUC
0	1	0.046332	0.046154	[0.968, 0.923, 0.952]	[0.969, 0.981, 0.965]
1	50	0.038610	0.061538	[0.933, 0.923, 0.955]	[0.938, 0.981, 0.977]
2	100	0.038610	0.046154	[0.951, 0.96, 0.955]	[0.953, 0.991, 0.977]
3	150	0.023166	0.015385	[0.984, 1.0, 0.977]	[0.984, 1.0, 0.989]
4	200	0.027027	0.015385	[0.984, 1.0, 0.977]	[0.984, 1.0, 0.989]
5	300	0.027027	0.015385	[0.984, 1.0, 0.977]	[0.984, 1.0, 0.989]
6	400	0.027027	0.015385	[0.984, 1.0, 0.977]	[0.984, 1.0, 0.989]

d)

In [103]:

```
rf_metrics = {}
rf_metrics["No of Trees"] = []
rf_metrics["Train Zero One Error"] = []
rf_metrics["Test Zero One Error"] = []
rf_metrics["F1 Score Train"] = []
rf_metrics["F1 Score Test"] = []
rf_metrics["AUC Train"] = []
rf_metrics["AUC Test"] = []

feature_imp = {}

for n_trees in [10,50,100]:

    rf_metrics["No of Trees"].append(n_trees)

    # Training a Random Forest Classifier
    clf = RandomForestClassifier(n_estimators=n_trees).fit(X_train_bootstrap, y_train_bootstrap.values.rave

    # Predicting the Classes
    y_pred_train = clf.predict(X_train)
    y_pred_test = clf.predict(X_test)

    # Computing the Zero One Test Error
    rf_metrics["Train Zero One Error"].append(1 - (np.array(y_train["species"]) == y_pred_train).sum()/len(
    rf_metrics["Test Zero One Error"].append(1 - (np.array(y_test["species"]) == y_pred_test).sum()/len(y_t

    # Computing the F1 Score
    f1score = f1_score(y_train["species"],y_pred_train,average=None)
    f1score = [round(x,3) for x in f1score]
    rf_metrics["F1 Score Train"].append(f1score)

    f1score = f1_score(y_test["species"],y_pred_test,average=None)
    f1score = [round(x,3) for x in f1score]
    rf_metrics["F1 Score Test"].append(f1score)

    # Computing the AUC - Train
    y_pred_class = pd.DataFrame(OneHotEncoder().fit_transform(y_pred_train.reshape(-1,1)).toarray()).astype
    y_train_class = pd.DataFrame(OneHotEncoder().fit_transform(y_train[["species"]]).toarray())

    auc_list = []
    for col1 in y_train_class.columns:
        auc = round(roc_auc_score(list(y_train_class[col1]),list(y_pred_class[col1])),3)
        auc_list.append(auc)

    rf_metrics["AUC Train"].append(auc_list)

    # Computing the AUC - Test
    y_pred_class = pd.DataFrame(OneHotEncoder().fit_transform(y_pred_test.reshape(-1,1)).toarray()).astype(
    y_test_class = pd.DataFrame(OneHotEncoder().fit_transform(y_test[["species"]]).toarray())

    auc_list = []
    for col1 in y_test_class.columns:
        auc = round(roc_auc_score(list(y_test_class[col1]),list(y_pred_class[col1])),3)
        auc_list.append(auc)

    rf_metrics["AUC Test"].append(auc_list)

    # Calculating the Top Ten Features wrt their importances
    imp_df = pd.DataFrame(clf.feature_importances_)
    imp_df.columns = ["Feature Importances"]
    imp_df["Features"] = list(X_train.columns)
    imp_df = imp_df[["Features","Feature Importances"]]

    top10_imp_df = imp_df.sort_values(["Feature Importances"],ascending=False).reset_index(drop=True)[:10]

    feature_imp[n_trees] = top10_imp_df.copy()

print("Reporting the Final Metrics for every tree(T): ")
pd.DataFrame(rf_metrics)
```

```
Reporting the Final Metrics for every tree(T):
```

Out[103]:

	No of Trees	Train Zero One Error	Test Zero One Error	F1 Score Train	F1 Score Test	AUC Train	AUC Test
0	10	0.023166	0.000000	[0.972, 0.945, 1.0]	[1.0, 1.0, 1.0]	[0.976, 0.965, 1.0]	[1.0, 1.0, 1.0]
1	50	0.015444	0.000000	[0.981, 0.964, 1.0]	[1.0, 1.0, 1.0]	[0.984, 0.977, 1.0]	[1.0, 1.0, 1.0]
2	100	0.019305	0.015385	[0.977, 0.955, 1.0]	[0.984, 1.0, 0.977]	[0.979, 0.974, 1.0]	[0.984, 1.0, 0.989]

In [104]:
```
for n_trees in feature_imp.keys():

    print(f"\nTop 10 Features wrt Feature Importances of a Random Forest Classifier of {n_trees} trees:")
    display(feature_imp[n_trees])
```

Top 10 Features wrt Feature Importances of a Random Forest Classifier of 10 trees:

	Features	Feature Importances
0	flipper_length_mm	0.218543
1	Delta 13 C (o/oo)	0.209097
2	culmen_depth_mm	0.188981
3	culmen_length_mm	0.144862
4	body_mass_g	0.097088
5	Delta 15 N (o/oo)	0.078094
6	island_Dream	0.055412
7	island_Torgersen	0.005335
8	sex_MALE	0.001599
9	studyName_PAL0910	0.000988

Top 10 Features wrt Feature Importances of a Random Forest Classifier of 50 trees:

	Features	Feature Importances
0	culmen_length_mm	0.306287
1	flipper_length_mm	0.150637
2	Delta 13 C (o/oo)	0.146349
3	body_mass_g	0.112374
4	Delta 15 N (o/oo)	0.096862
5	culmen_depth_mm	0.096744
6	island_Dream	0.071028
7	studyName_PAL0809	0.006501
8	studyName_PAL0910	0.004701
9	sex_MALE	0.003870

Top 10 Features wrt Feature Importances of a Random Forest Classifier of 100 trees:

	Features	Feature Importances
0	culmen_length_mm	0.239834
1	flipper_length_mm	0.232335
2	Delta 13 C (o/oo)	0.175041
3	body_mass_g	0.109158
4	culmen_depth_mm	0.106433
5	Delta 15 N (o/oo)	0.067624
6	island_Dream	0.048551
7	studyName_PAL0809	0.006786
8	studyName_PAL0910	0.006202
9	sex_MALE	0.004824

Problem 5

a)

Solution:

Let h1 and h2 represent the nodes in the hidden layer,

From the given neural network architecture:

$$h_1 = c(w_1 + w_3 . x_1 + w_5 . x_2)$$

$$h_2 = c(w_2 + w_4 . x_1 + w_6 . x_2)$$

Let z represent the value of the output node before applying the activation function,

$$z = w_7 + h_1 . w_8 + h_2 . w_9$$

$$z = w_7 + c . w_8 . (w_1 + w_3 . x_1 + w_5 . x_2) + c . w_9 (w_2 + w_4 . x_1 + w_6 . x_2)$$

$$z = (w_7 + c . w_8 . w_1 + c . w_9 . w_2) + (c . w_8 . w_3 + c . w_9 . w_4) . x_1 + (c . w_8 . w_5 + c . w_9 . w_6) . x_2$$

Now applying the activation function (sigmoid), we get:

$$P(y = 1) = \frac{1}{(1+e^{-z})}$$

$$\Rightarrow P(y = 1) = \frac{1}{(1+e^{-((w_7+c.w_8.w_1+c.w_9.w_2)+(c.w_8.w_3+c.w_9.w_4).x_1+(c.w_8.w_5+c.w_9.w_6).x_2)})}$$

The final classification boundary is when:

P(y=1) = 0.5

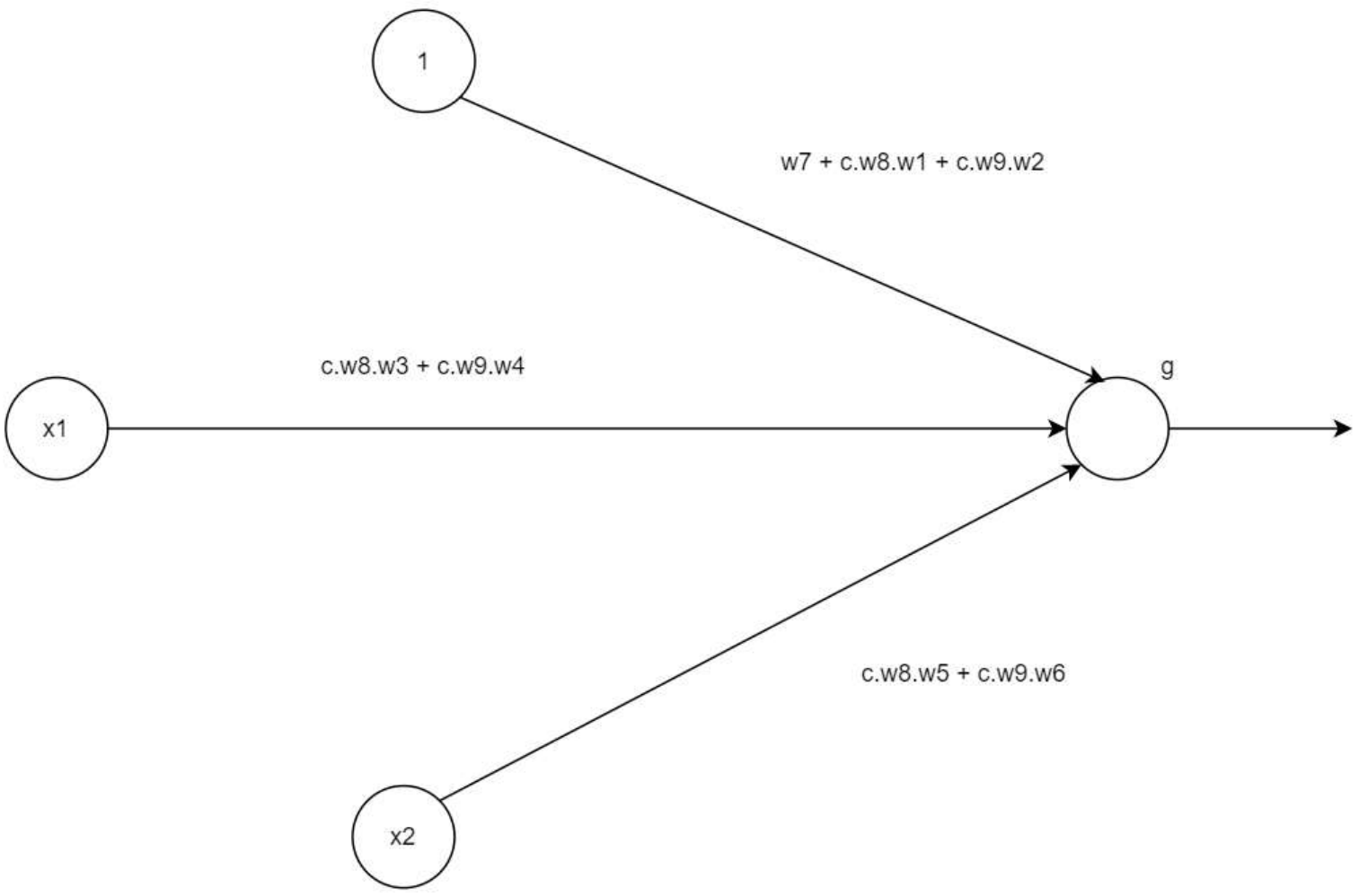
$$\Rightarrow \frac{1}{(1+e^{-z})} = 0.5 \Rightarrow z = 0$$

Therefore, the classification boundary is:

$$(w_7 + c . w_8 . w_1 + c . w_9 . w_2) + (c . w_8 . w_3 + c . w_9 w_4) . x_1 + (c . w_8 . w_5 + c . w_9 . w_6) . x_2 = 0$$

We obtain a linear classification boundary in terms of x

b)



Let the new weights be $w_{1,} w_2, w_3$

$$w_1 = w_7 + c . w_8 . w_1 + c . w_9 . w_2$$

$$w_2 = c . w_8 . w_3 + c . w_9 . w_4$$

$$w_3 = c . w_{.8} . w_5 + c . w_9 . w_6$$

$$P(y = 1) = g(w_1 + w_2 . x_1 + w_3 . x_2)$$

Yes. If linear activation functions are used for all the hidden units, output from hidden units will be written as linear combination of input features. Since these intermediate output serves as input for the final output layer, we can always find an equivalent neural net which does not have any hidden layer as seen in the example above.