Gemini

Required Packages

```
import pandas as pd
import numpy
import string
from sklearn.model_selection import train_test_split
import sklearn.decomposition
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
%pylab inline
import math
import statistics
import sklearn
from sklearn import neighbors
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, completeness_score
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn import tree
import graphviz
import pydot
import pydotplus
from sklearn import svm
import os
os.environ["PATH"] += os.pathsep + "C:\\Program Files (x86)\\Graphviz2.38\\bin\\"
import seaborn as sns
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Load the Data

```
#Original Data
fraud = pd.read_csv('/content/drive/My Drive/fraudData.csv')
#Upsampled Data - Training


df2=pd.read_csv('/content/drive/My Drive/creditcard_new.csv')
df2.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0

5 rows × 31 columns

```
#checking correlation
corr=df2.corr()
corr
```

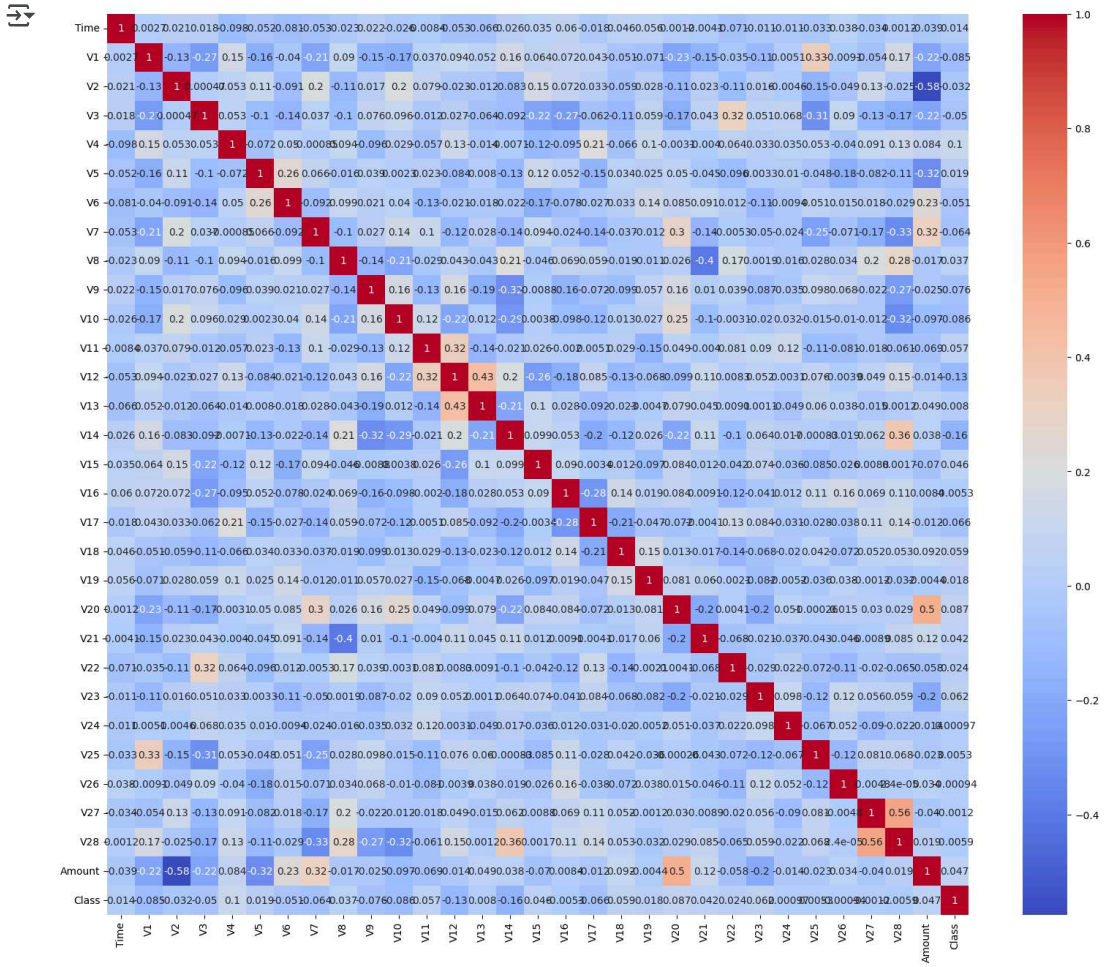
1 to 25 of 31 entries Filter

V11	V12	V13	V14	V15	
)8437835780781083	-0.052575865163924575	-0.06641950293511927	0.025964542148318545	0.034761453155989164	0.05
)3654766472881237	0.09426707334833272	0.05198081121322252	0.1562388884492963	0.06423869202142417	0.07
)7922125192744223	-0.022688765259381027	-0.012433534998105617	-0.08295826010663548	0.1485132744976339	0.(
11860191200010141	0.02712333809918995	-0.06413936559662822	-0.09157601056294197	-0.22258537614632276	-0.2
)5748709360756093	0.12865793177929444	-0.013502573777133736	-0.007090143501854 0715	-0.12164096157552506	-0.09
:3275120944077264	-0.08402449241020332	0.008024353553920402	-0.12701046237748517	0.12199209790956206	0.05
13037123879899884	-0.021358919198941218	-0.017882737058407365	-0.021789479667503164	-0.16647797623433316	-0.07
.1038340437015405	-0.1237389127533126	0.028168863557450498	-0.1371525385044394	0.09401130768383535	-0.02
:8751597633427543	0.042776208139140515	-0.042546003567187386	0.21106113066563886	-0.04554745821287145	0.06
12530934348670555	0.1631940728336848	-0.18533603945754465	-0.31734383666900445	-0.008778201842439791	-0.15
12273976228930346	-0.22314919961394414	0.012231979070942249	-0.2897049532019623	0.003773063113769182	-0.09
1.0	0.3242600544001402	-0.13935926033255175	-0.02107317133595728	0.026409604289295924	-0.00
.3242600544001402	1.0	0.43176493735916455	0.19579824728674475	-0.26057021294792854	-0.18
13935926033255175	0.43176493735916455	1.0	-0.20634096333381025	0.09961072686167115	0.02
)2107317133595728	0.19579824728674475	-0.20634096333381025	1.0	0.0993469629252637	0.05
:6409604289295924	-0.26057021294792854	0.09961072686167115	0.0993469629252637	1.0	0.08
)1981310969731231	-0.18323339414605105	0.027726472236011938	0.05320767600803998	0.08990185559599198	
51321155182635384	0.08512141166786617	-0.09178227150640092	-0.19612874199337108	-0.003439086971153879	-0.27
)2949286895120329	-0.12771106616577563	-0.022750317733714366	-0.12422829803207894	0.011821863928408624	0.14
14829287482708423	-0.06785939127909227	-0.004685302100674802	0.02556348971953643	-0.09668312057333205	0.01
)4872634407462445	-0.0994251758193897	0.07925103188033811	-0.2205955000149275	0.0839192453520951	0.08
)4046093130136478	0.1098529288810359	0.04499512936672348	0.11274642059652253	0.011688002601482225	0.009
)8122283598427922	0.008292519065747366	0.009108774363747824	-0.10176283501218654	-0.04232615802377164	-0.12
)8962406223521946	0.05165309175615279	0.001134948150927253	0.06372415235131809	0.0744857090369862	-0.04
12438156752530173	0.003123107526134647	-0.0487036695032196	0.01663622552945568	-0.03593423743112617	0.012

Show 25 per page 1 2

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
#checking the correlation in hetmap
plt.figure(figsize=(19,16))
sns.heatmap(corr, cmap="coolwarm",annot=True)
plt.show()
```

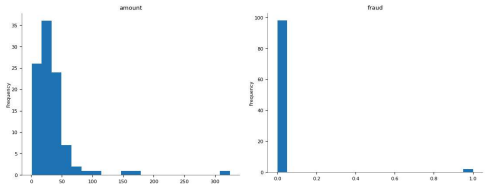



```
df=pd.DataFrame(fraud)
df.head(100)
```

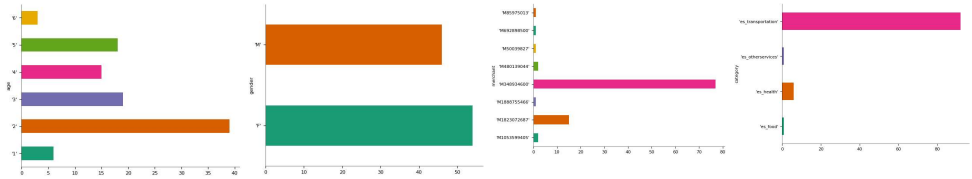
	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fr
0	0	'C1093826151'	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4.55	
1	0	'C352968107'	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	39.68	
2	0	'C2054744914'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.89	
3	0	'C1760612790'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	17.25	
4	0	'C757503768'	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	35.72	
...	...	...	...	...	...	...	...	...	...	
95	0	'C1697851479'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	20.73	
96	0	'C1255236689'	'1'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	68.17	
97	0	'C603081336'	'3'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	34.75	
98	0	'C274486575'	'2'	'F'	'28007'	'M692898500'	'28007'	'es_health'	171.07	
99	0	'C1569939854'	'6'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	50.31	

100 rows × 10 columns

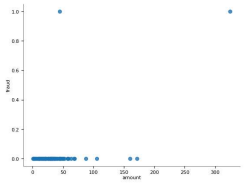
Distributions



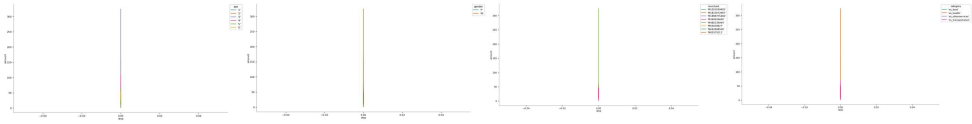
Categorical distributions



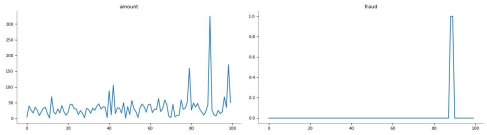
2-d distributions



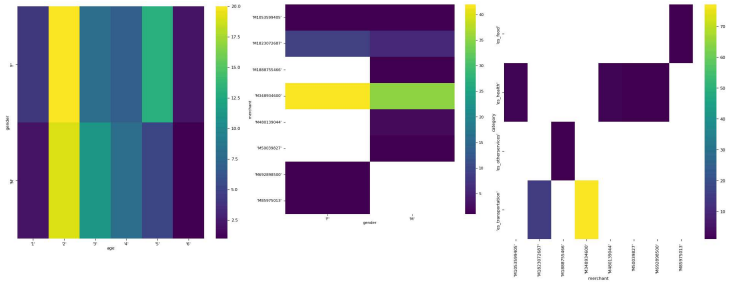
Time series



Values



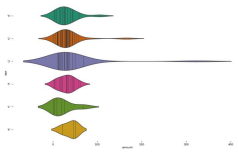
2-d categorical distributions



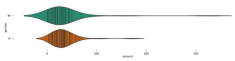
Faceted distributions

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x`
```

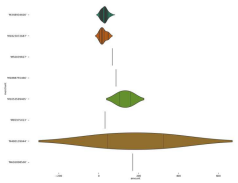
```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y`
```



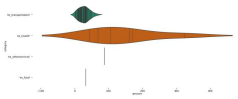
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y`
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y`
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y`
```



Pre-process the Original Data

[ ] ↳ 10 cells hidden

Standardize the Data for Customer Clustering

Original Data

```
scaler = StandardScaler()
fraudElim = fraudBinaryCols
fraudElim = fraudElim.drop(["step", "fraud"], axis = 1)
fraudStandScaler = scaler.fit_transform(fraudBinaryCols)
fraudStand = pandas.DataFrame(fraudStandScaler)
fraudStand.columns = list(fraudBinaryCols)
dataStandardized = fraudStand
dataStandardized["fraud"] = fraudBinaryCols["fraud"]
```

Upsampled Data

```
scaler = StandardScaler()
fraudStandScaler = scaler.fit_transform(fraudUp)
fraudStand = pandas.DataFrame(fraudStandScaler)
fraudStand.columns = list(fraudBinaryCols)
dataStandardizedUP = fraudStand
```

Test Data (from Upsampled)

```
fraudTestTest = fraudTest
fraudTestTest = fraudTestTest.drop("fraud", axis = 1)
scaler = StandardScaler()
fraudStandScaler = scaler.fit_transform(fraudTestTest)
fraudStand = pandas.DataFrame(fraudStandScaler)
fraudStand.columns = list(fraudTestTest)
dataStandardizedTest = fraudStand
```

Attempt to Cluster Customers by Purchase Behavior

Original Data - Standardized

K-Means Clustering

```
error = list()
kList = list()
for k in range(2, 11):
    kmeans_model = KMeans(n_clusters = k, random_state = 891).fit(dataStandardized)
    labels = kmeans_model.labels_
    labels = labels.tolist()
    cost = kmeans_model.inertia_
    error.append(cost)
    kList.append(k)
    print("k:", k, " cost:", cost)
plt.plot(kList, error)
plt.title("K Values And Error Score - Original Data")
plt.xlabel("K Value")
plt.ylabel("Error Score")
plt.grid()
plt.show()
```

```
k: 2  cost: 45582627.414760284
k: 3  cost: 44082528.24596801
k: 4  cost: 43113145.498852685
k: 5  cost: 41905419.66589378
k: 6  cost: 40600675.23316435
k: 7  cost: 39879098.315167114
k: 8  cost: 39451977.14252325
k: 9  cost: 38151369.41422901
k: 10  cost: 37548416.16936904
```

Using the Best Value of k

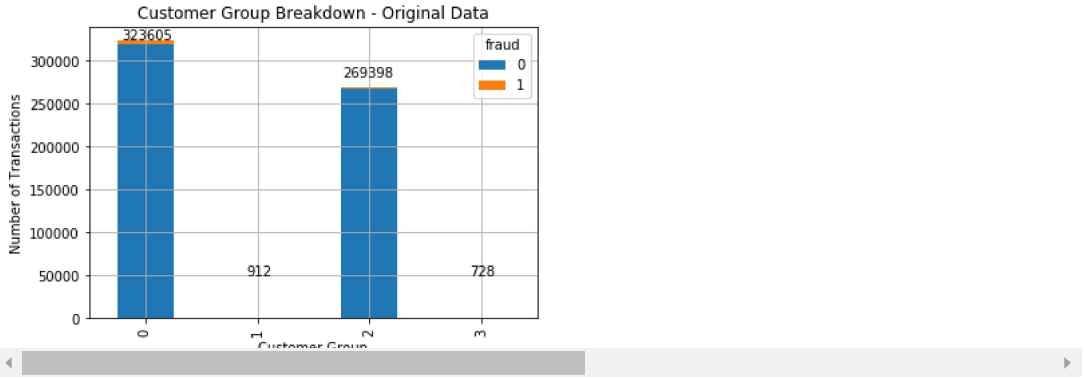
```
kmeans = KMeans(n_clusters = 4, random_state = 891).fit(dataStandardized)
labs = kmeans.labels_
labsList = labs.tolist()
dataStandardized["customerGroup"] = labsList
dataStandardized["fraud"] = fraud["fraud"]
```

Examine Customer Group Distribution

```
counts = dataStandardized.groupby("customerGroup").count()
counts = counts["step"].values.tolist()

dataStandardized.groupby(['customerGroup', 'fraud']).size().unstack().plot(kind='bar', stacked=True)
annotate(counts[0], [-0.21, 325000])
annotate(counts[1], [0.9, 50000])
annotate(counts[2], [1.76, 280000])
annotate(counts[3], [2.9, 50000])

ylabel("Number of Transactions")
xlabel("Customer Group")
```

```
grid()
title("Customer Group Breakdown - Original Data")
```

```
Text(0.5,1,'Customer Group Breakdown - Original Data')
```




```
cust0 = dataStandardized.loc[dataStandardized['customerGroup'] == 0]
cust0 = cust0.reset_index(drop = True)
cust1 = dataStandardized.loc[dataStandardized['customerGroup'] == 1]
cust1 = cust1.reset_index(drop = True)
cust2 = dataStandardized.loc[dataStandardized['customerGroup'] == 2]
cust2 = cust2.reset_index(drop = True)
cust3 = dataStandardized.loc[dataStandardized['customerGroup'] == 3]
cust3 = cust3.reset_index(drop = True)

counts0 = cust0.groupby("fraud").count()
counts0 = counts0["step"].values.tolist()
nofraud0 = counts0[0]/sum(counts0)
fraud0 = counts0[1]/sum(counts0)
print("Customer Group 0:", "%s not fraud" % nofraud0, "| %s fraud" % fraud0)

counts1 = cust1.groupby("fraud").count()
counts1 = counts1["step"].values.tolist()
nofraud1 = counts1[0]/sum(counts1)
fraud1 = counts1[1]/sum(counts1)
print("Customer Group 1:", "%s not fraud" % nofraud1, "| %s fraud" % fraud1)

counts2 = cust2.groupby("fraud").count()
counts2 = counts2["step"].values.tolist()
nofraud2 = counts2[0]/sum(counts2)
fraud2 = counts2[1]/sum(counts2)
print("Customer Group 2:", "%s not fraud" % nofraud2, "| %s fraud" % fraud2)

counts3 = cust3.groupby("fraud").count()
counts3 = counts3["step"].values.tolist()
nofraud3 = counts3[0]/sum(counts3)
fraud3 = counts3[1]/sum(counts3)
print("Customer Group 3:", "%s not fraud" % nofraud3, "| %s fraud" % fraud3)
```

```
Customer Group 0: 0.9869006968371935 not fraud | 0.013099303162806508 fraud
Customer Group 1: 0.75 not fraud | 0.25 fraud
Customer Group 2: 0.9920006830043282 not fraud | 0.00799931699567183 fraud
Customer Group 3: 0.20604395604395603 not fraud | 0.7939560439560439 fraud
```

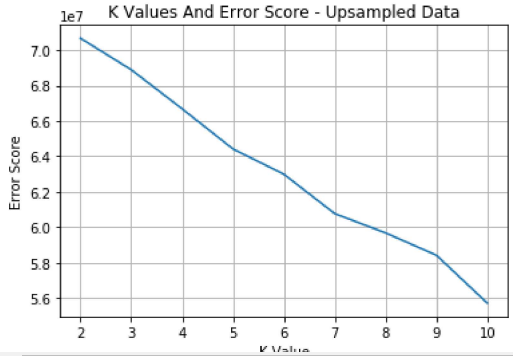
Upsampled Data - Standardized

K-Means Clustering

```
error = list()
kList = list()
for k in range(2, 11):
    kmeans_model = KMeans(n_clusters = k, random_state = 891).fit(dataStandardizedUP)
    labels = kmeans_model.labels_
    labels = labels.tolist()
    cost = kmeans_model.inertia_
    error.append(cost)
    kList.append(k)
    print("k:", k, " cost:", cost)
plot(kList, error)
```

```
title("K Values And Error Score - Upsampled Data")
xlabel("K Value")
ylabel("Error Score")
grid()
show()
```

```
k: 2  cost: 70679353.74244852
k: 3  cost: 68894406.45436603
k: 4  cost: 66690476.96307046
k: 5  cost: 64420003.92912575
k: 6  cost: 62993960.50975509
k: 7  cost: 60770439.98079894
k: 8  cost: 59677998.21574084
k: 9  cost: 58416067.61473298
k: 10  cost: 55712832.18699259
```



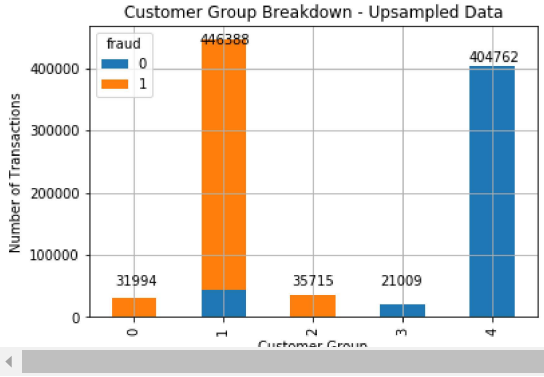
Using the Best Value of k

```
kmeans = KMeans(n_clusters = 5, random_state = 891).fit(dataStandardizedUP)
labs = kmeans.labels_
labsList = labs.tolist()
dataStandardizedUP["customerGroup"] = labsList
dataStandardizedUP["fraud"] = fraudUp["fraud"]
```

Examine Customer Group Distribution

```
counts = dataStandardizedUP.groupby("customerGroup").count()
counts = counts["amount"].values.tolist()

dataStandardizedUP.groupby(['customerGroup', 'fraud']).size().unstack().plot(kind='bar', stacked=True)
annotate(counts[0], [-0.21, 50000])
annotate(counts[1], [0.73, 439000])
annotate(counts[2], [1.76, 50000])
annotate(counts[3], [2.74, 50000])
annotate(counts[4], [3.73, 410000])

ylabel("Number of Transactions")
xlabel("Customer Group")
grid()
title("Customer Group Breakdown - Upsampled Data")
```

```
Text(0.5,1,'Customer Group Breakdown - Upsampled Data')
```



Gemini

Gemini is a powerful AI tool built by Google that helps you to use Colab.
Not sure what to ask?
Try a suggested prompt below

How do I filter a Pandas DataFrame?

How can I create a plot in Colab?

Show me a list of publicly available datasets

```
cust0 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 0]
cust0 = cust0.reset_index(drop = True)
cust1 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 1]
cust1 = cust1.reset_index(drop = True)
cust2 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 2]
cust2 = cust2.reset_index(drop = True)
cust3 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 3]
cust3 = cust3.reset_index(drop = True)
cust4 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 4]
cust4 = cust4.reset_index(drop = True)

counts0 = cust0.groupby("fraud").count()
counts0 = counts0["amount"].values.tolist()
nofraud0 = counts0[0]/sum(counts0)
fraud0 = counts0[1]/sum(counts0)
print("Customer Group 0:", "%s not fraud" % nofraud0, "| %s fraud" % fraud0)

counts1 = cust1.groupby("fraud").count()
counts1 = counts1["amount"].values.tolist()
nofraud1 = counts1[0]/sum(counts1)
fraud1 = counts1[1]/sum(counts1)
print("Customer Group 1:", "%s not fraud" % nofraud1, "| %s fraud" % fraud1)

counts2 = cust2.groupby("fraud").count()
counts2 = counts2["amount"].values.tolist()
nofraud2 = counts2[0]/sum(counts2)
fraud2 = counts2[1]/sum(counts2)
print("Customer Group 2:", "%s not fraud" % nofraud2, "| %s fraud" % fraud2)

counts3 = cust3.groupby("fraud").count()
counts3 = counts3["amount"].values.tolist()
nofraud3 = counts3[0]/sum(counts3)
print("Customer Group 3:", "%s not fraud" % nofraud3, "| %s fraud" % 0)

counts4 = cust4.groupby("fraud").count()
counts4 = counts4["amount"].values.tolist()
nofraud4 = counts4[0]/sum(counts4)
print("Customer Group 4:", "%s not fraud" % nofraud4, "| %s fraud" % 0)
```

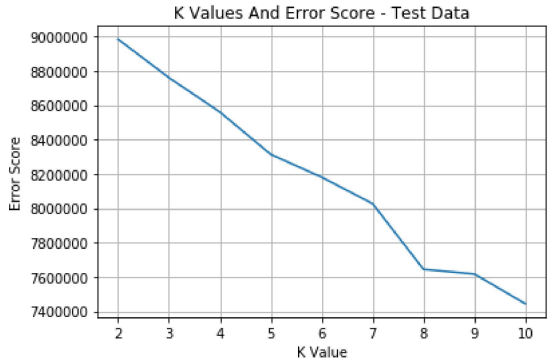
```
Customer Group 0: 0.0006251172094767769 not fraud | 0.9993748827905232 fraud
Customer Group 1: 0.09862720324023047 not fraud | 0.9013727967597696 fraud
Customer Group 2: 0.003275934481310374 not fraud | 0.9967240655186896 fraud
Customer Group 3: 1.0 not fraud | 0 fraud
Customer Group 4: 1.0 not fraud | 0 fraud
```

Test Data - Standardized

K-Means Clustering

```
error = list()
kList = list()
for k in range(2, 11):
    kmeans_model = KMeans(n_clusters = k, random_state = 891).fit(dataStandardizedTest)
    labels = kmeans_model.labels_
    labels = labels.tolist()
    cost = kmeans_model.inertia_
    error.append(cost)
    kList.append(k)
    print("k:", k, " cost:", cost)
plot(kList, error)
title("K Values And Error Score - Test Data")
xlabel("K Value")
ylabel("Error Score")
grid()
show()
```

```
k: 2  cost: 8983754.79033209
k: 3  cost: 8759730.472061029
k: 4  cost: 8560138.352494117
k: 5  cost: 8313444.254985454
k: 6  cost: 8180582.2624179255
k: 7  cost: 8026945.636027608
k: 8  cost: 7644403.490412499
k: 9  cost: 7617370.704876197
k: 10  cost: 7444527.622420674
```



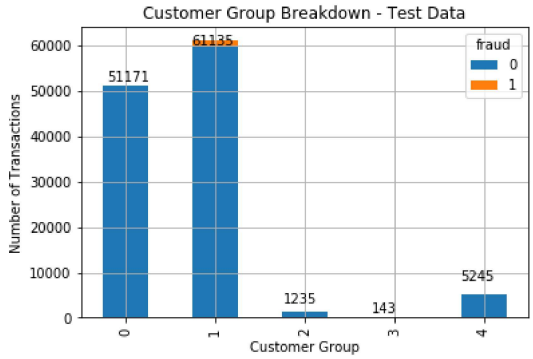
Using the Best Value of k

```
kmeans = KMeans(n_clusters = 5, random_state = 891).fit(dataStandardizedTest)
labs = kmeans.labels_
labsList = labs.tolist()
dataStandardizedTest["customerGroup"] = labsList
dataStandardizedTest["fraud"] = fraudTest["fraud"]
```

Examine Customer Group Distribution

```
counts = dataStandardizedTest.groupby("customerGroup").count()
counts = counts["amount"].values.tolist()

dataStandardizedTest.groupby(['customerGroup', 'fraud']).size().unstack().plot(kind='bar', stacked=True)
annotate(counts[0], [-0.21, 52000])
annotate(counts[1], [0.73, 60000])
annotate(counts[2], [1.76, 3000])
annotate(counts[3], [2.74, 1000])
annotate(counts[4], [3.73, 8200])

ylabel("Number of Transactions")
xlabel("Customer Group")
grid()
title("Customer Group Breakdown - Test Data")
```

```
Text(0.5,1,'Customer Group Breakdown - Test Data')
```




```
cust0 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 0]
cust0 = cust0.reset_index(drop = True)
cust1 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 1]
cust1 = cust1.reset_index(drop = True)
cust2 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 2]
cust2 = cust2.reset_index(drop = True)
cust3 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 3]
```

```
cust3 = cust3.reset_index(drop = True)
cust4 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 4]
cust4 = cust4.reset_index(drop = True)

counts0 = cust0.groupby("fraud").count()
counts0 = counts0["amount"].values.tolist()
nofraud0 = counts0[0]/sum(counts0)
print("Customer Group 0:", "%s not fraud" % nofraud0)#, "%s fraud" % fraud0)

counts1 = cust1.groupby("fraud").count()
counts1 = counts1["amount"].values.tolist()
nofraud1 = counts1[0]/sum(counts1)
fraud1 = counts1[1]/sum(counts1)
print("Customer Group 1:", "%s not fraud" % nofraud1, "| %s fraud" % fraud1)

counts2 = cust2.groupby("fraud").count()
counts2 = counts2["amount"].values.tolist()
nofraud2 = counts2[0]/sum(counts2)
fraud2 = counts2[1]/sum(counts2)
print("Customer Group 2:", "%s not fraud" % nofraud2, "| %s fraud" % fraud2)

counts3 = cust3.groupby("fraud").count()
counts3 = counts3["amount"].values.tolist()
nofraud3 = counts3[0]/sum(counts3)
fraud3 = counts3[1]/sum(counts3)
print("Customer Group 3:", "%s not fraud" % nofraud3, "| %s fraud" % fraud3)

counts4 = cust4.groupby("fraud").count()
counts4 = counts4["amount"].values.tolist()
nofraud4 = counts4[0]/sum(counts4)
print("Customer Group 4:", "%s not fraud" % nofraud4)#, "%s fraud" % fraud4)
```

```
Customer Group 0: 1.0 not fraud
Customer Group 1: 0.9791281589923939 not fraud | 0.02087184100760612 fraud
Customer Group 2: 0.9748987854251012 not fraud | 0.025101214574898785 fraud
Customer Group 3: 0.2097902097902098 not fraud | 0.7902097902097902 fraud
Customer Group 4: 1.0 not fraud
```

Split Data into Train Test

```
dataStandardized1 = dataStandardized
dataStandardized1 = dataStandardized1.drop("fraud", axis = 1)
X_train, X_test, y_train, y_test = train_test_split(dataStandardized1, dataStandardized['fraud'], test_size =
```

Naive Bayes

```
gnb = GaussianNB()
nb = cross_val_score(gnb, X_train, y_train, cv = 10)
print("Train Data:", numpy.mean(nb))

gnb = GaussianNB()
nb = cross_val_score(gnb, X_test, y_test, cv = 10)
print("Test Data:", numpy.mean(nb))
```

```
Train Data: 0.9507042088251954
Test Data: 0.9501341129450445
```

Logistic Regression

```
reg = sklearn.linear_model.LogisticRegression()
reg.fit(X_train, y_train)
print(reg.coef_)
y_pred = reg.predict(X_test)
confMat = sklearn.metrics.confusion_matrix(y_test, y_pred)
confMatList = confMat.tolist()
TN = confMatList[0][0]
TP = confMatList[1][1]
FN = confMatList[1][0]
FP = confMatList[0][1]
```

```
precision = (TP) / (TP + FP)
recall = (TP) / (TP + FN)
print("Precision:", precision)
print("Recall:", recall)
confMat
```

```
[[ 0.34564236  1.45480284 -0.06327624  0.01653141  0.00553531 -0.02211753
   0.01278538  0.01781141 -0.0161141  -0.0214958  -0.0214958   0.09324961
  -0.08612507 -0.08888809 -0.28946347 -0.01800635  0.294898    0.098587
  -0.02336559 -0.08820967  0.08558719 -0.11031536 -0.06295951  0.26915115
   0.26849323 -0.09250767  0.07869496 -0.03713408  0.09439651  0.10722117
   0.10784619 -0.09594644 -0.16007575 -0.12007918 -0.09326862  0.16978337
   0.13507215 -0.22846964 -0.27896592  0.13845099  0.01721911  0.29293434
   0.17480885  0.0596751  -0.06294638 -0.1013012   0.1081484  -0.11461935
   0.54606914 -0.13131509  0.15851804  0.26758384 -0.01483232  0.12059642
   0.06384526  0.18116047  0.20074922  0.30185636  0.10018849 -0.13240982
   0.15088436  0.02576749 -0.09346524  0.31070779  0.26915115 -0.02747386
  -0.02397785 -0.13240982  0.02001663  0.01457554  0.06923648  0.30185636
   0.14598932  0.13507215  0.29679227  0.06745923 -0.30750631  0.01994153
   0.22909168]]
Precision: 0.8886993603411514
Recall: 0.7236111111111111
array([[234717,    261],
       [   796,   2084]], dtype=int64)
```

Neural Network

```
reg = MLPRegressor()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
y_pred = numpy.rint(y_pred)

confMat = sklearn.metrics.confusion_matrix(y_test, y_pred)
confMatList = confMat.tolist()
TN = confMatList[0][0]
TP = confMatList[1][1]
FN = confMatList[1][0]
FP = confMatList[0][1]

precision = (TP) / (TP + FP)
recall = (TP) / (TP + FN)
print("Precision:", precision)
print("Recall:", recall)
confMat
```

```
Precision: 0.8720699245133094
Recall: 0.7624175060784995
array([[234656,    322,      0],
       [   684,   2195,      1],
       [     0,      0,      0]], dtype=int64)
```

Decision Tree - Different Max_Depth Values

Determine Best Max_Depth Value

```
accuracy = []
for x in range(2, 101):
    print(x)
    clf = tree.DecisionTreeClassifier(max_depth = x)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    confMat = sklearn.metrics.confusion_matrix(y_test, y_pred)
    confMatList = confMat.tolist()
    TN = confMatList[0][0]
    TP = confMatList[1][1]
    FN = confMatList[1][0]
    FP = confMatList[0][1]

    precision = (TP) / (TP + FP)
    recall = (TP) / (TP + FN)
    f1 = 2*((precision * recall) / (precision + recall))
```