Sarah Riaz

sriaz@usc.edu

1389209718

Homework 3

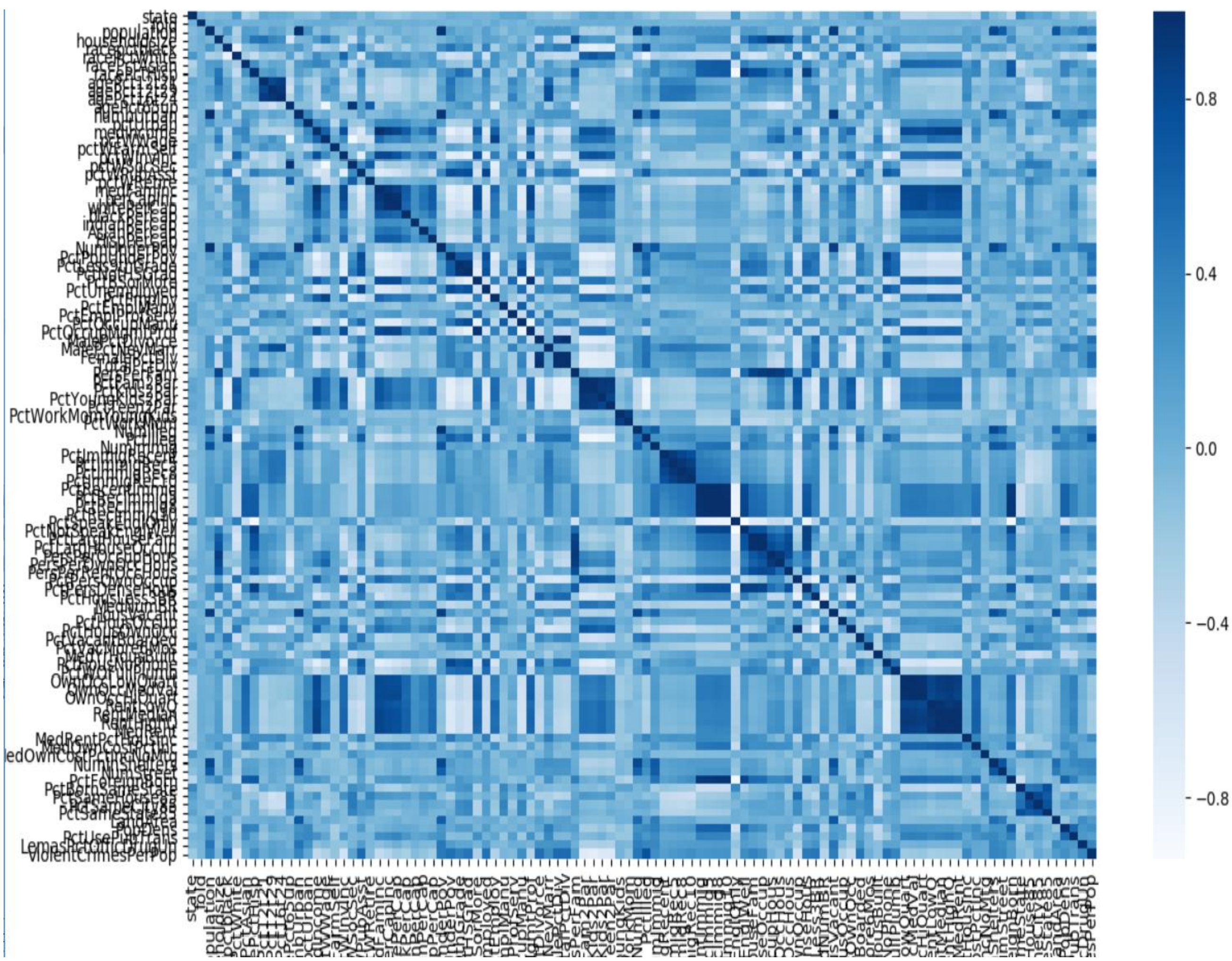
Question 1:

b:

I imputed the missing value '?' by zero.

c:

Correlation matrix for the features:



Code Snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt

Correlations =data_frame.corr()
plt.figure(figsize=(100,100))
sns.heatmap(correlations,cmap="Blues")
plt.show()
```

d:

Coefficient of Variation for each feature = **Standard Variation/Mean**

Coefficient of Variation for each feature are as follows:

state	0.571671
fold	0.523062
population	2.203503
householdsize	0.353298
racepctblack	1.410920
racePctWhite	0.323782
racePctAsian	1.359162
racePctHisp	1.614278
agePct12t21	0.365840
agePct12t29	0.290693
agePct16t24	0.495161
agePct65up	0.423442
numbUrban	2.001744
pctUrban	0.638849
medIncome	0.579753
pctWWage	0.327710
pctWFarmSelf	0.700030
pctWInvInc	0.359240
pctWSocSec	0.368513
pctWPubAsst	0.699031

pctWRetire	0.349639
medFamInc	0.527732
perCapInc	0.545633
whitePerCap	0.507552
blackPerCap	0.589469
indianPerCap	0.809685
AsianPerCap	0.606194
HispPerCap	0.473960
NumUnderPov	2.304970
PctPopUnderPov	0.753980
...	
HousVacant	1.958780
PctHousOccup	0.269647
PctHousOwnOcc	0.337541
PctVacantBoarded	1.064742
PctVacMore6Mos	0.436119
MedYrHousBuilt	0.470411
PctHousNoPhone	0.918211
PctWOFullPlumb	0.848744
OwnOccLowQuart	0.847880
OwnOccMedVal	0.878750
OwnOccHiQuart	0.874733
RentLowQ	0.633186
RentMedian	0.561884
RentHighQ	0.587014
MedRent	0.555592
MedRentPctHousInc	0.345830
MedOwnCostPctInc	0.416391
MedOwnCostPctIncNoMtg	0.476933

NumInShelters 3.485481

NumStreet 4.407702

PctForeignBorn 1.072291

PctBornSameState 0.335575

PctSameHouse85 0.338944

PctSameCity85 0.320105

PctSameState85 0.304240

LandArea 1.678031

PopDens 0.872187

PctUsePubTrans 1.416673

LemasPctOfficDrugUn 2.555266

ViolentCrimesPerPop 0.979015

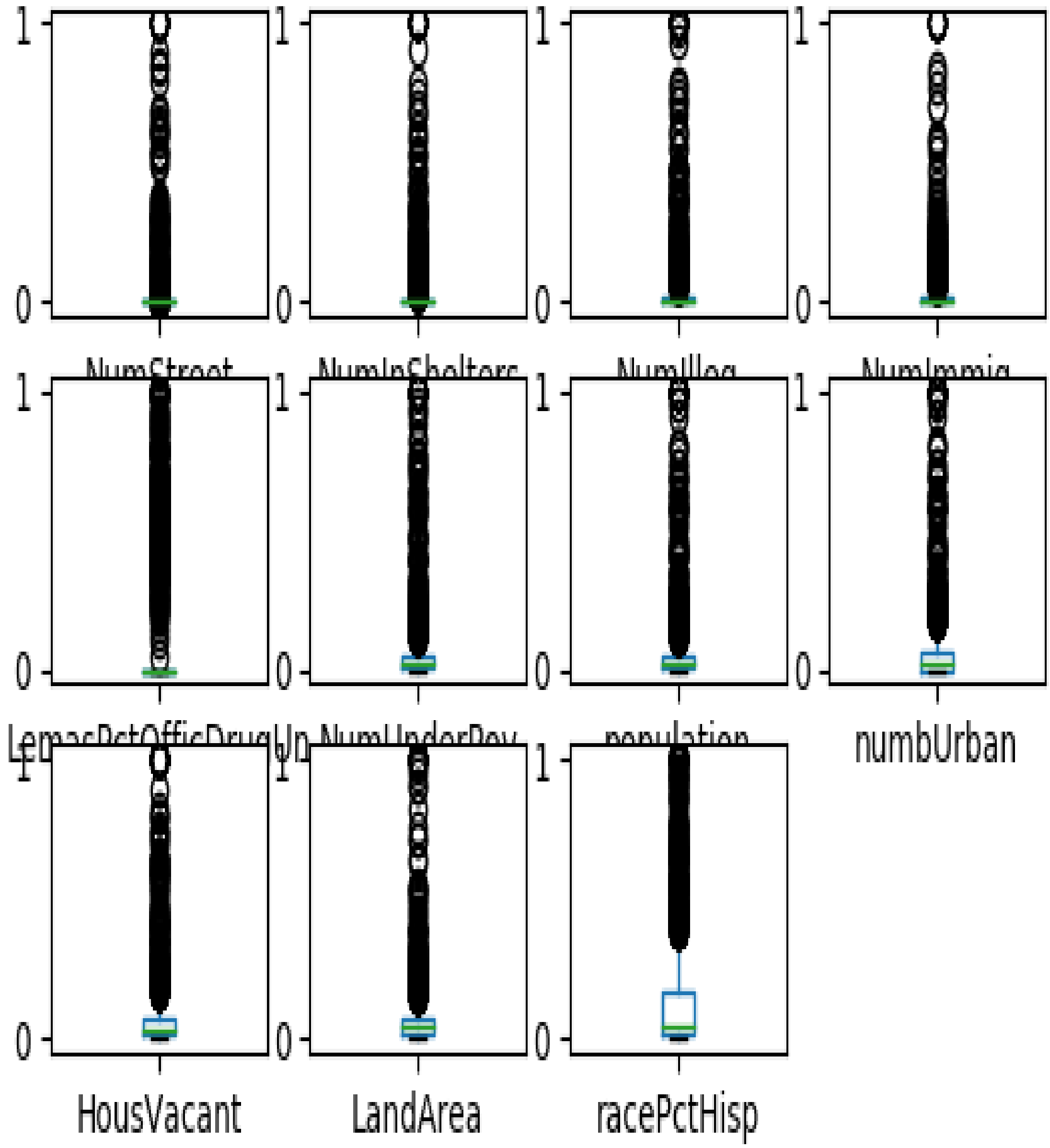
Code Snippet:

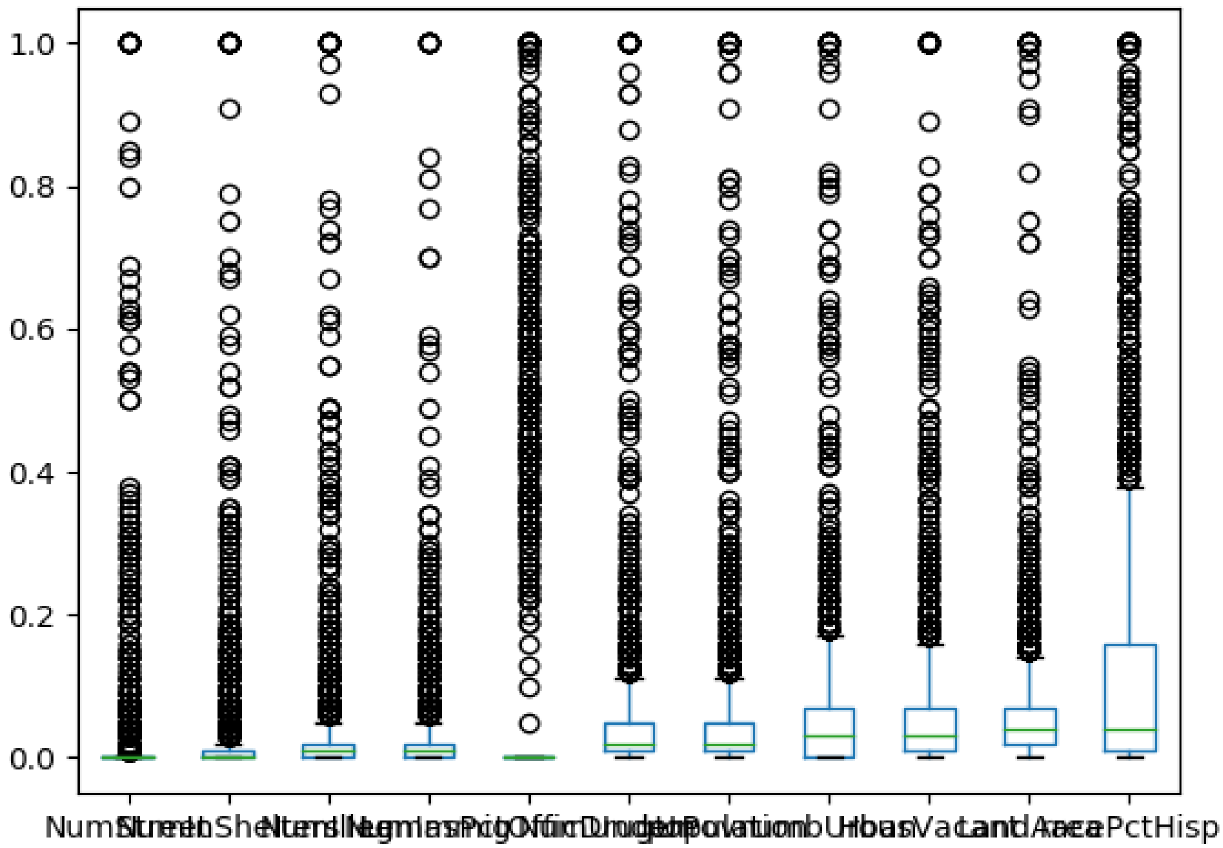
```
import pandas as pd

df=pd.read_csv("C:/Users/Sarah Riaz/Documents/ML/HW/HW3/Crime-data/Crime-
data/communities.csv")

CV=df.std()/df.mean()
print(CV)
```

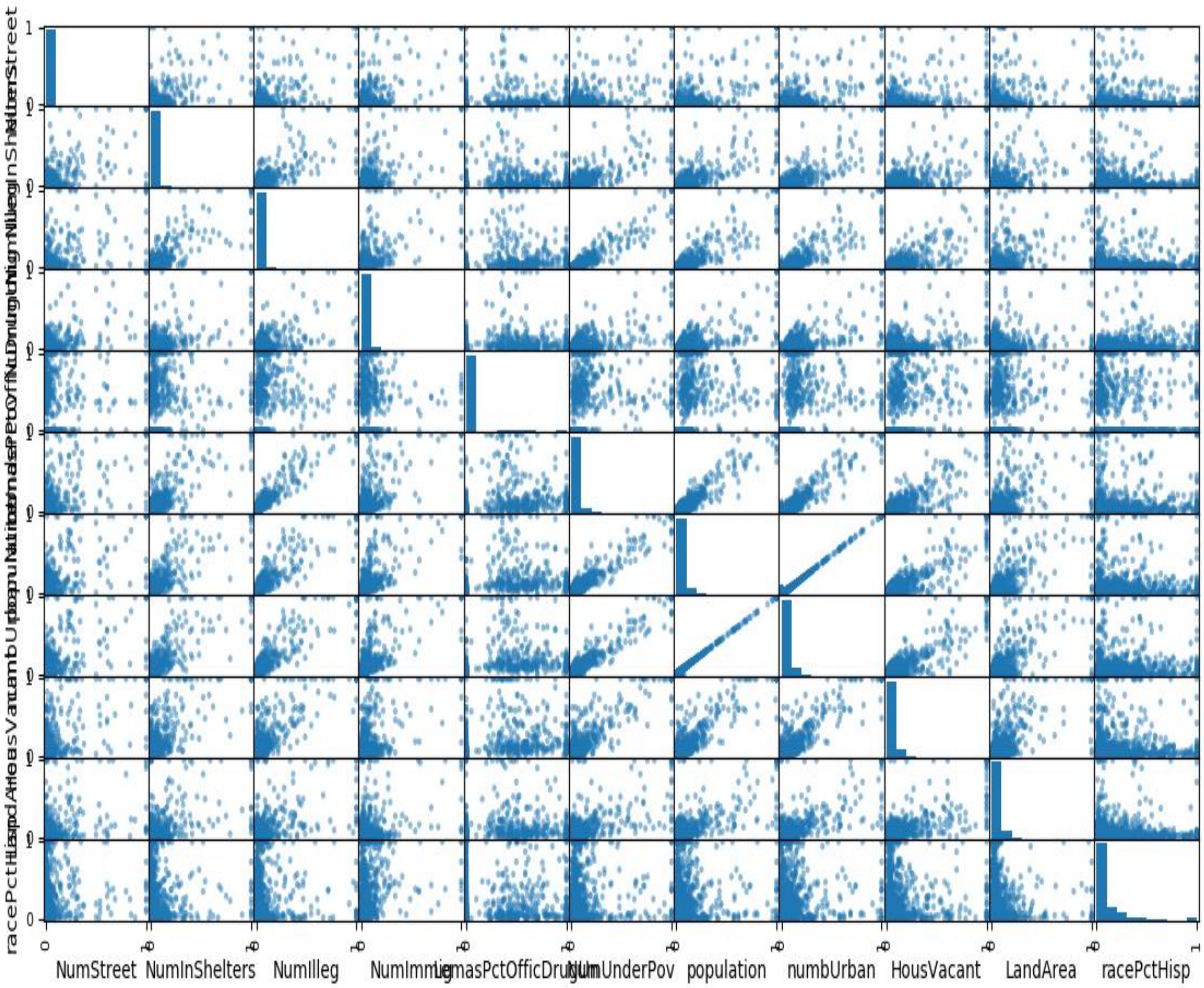
e:

√128 features with highest CV:
'NumStreet', 'NumInShelters', 'NumIlleg', 'NumImmig', 'LemasPctOfficDrugUn', 'NumUnderPov', 'numbUrban', 'population', 'LandArea', 'racePctHisp’, and 'HousVacant'

Scatter and box plot are as follows:


1
0
NumStreet
NumInShelters
NumIlleg
NumImmig
LemasPctOfficDrugUn
NumUnderPov
population
numbUrban
HousVacant
LandArea
racePctHisp


1.0
0.8
0.6
0.4
0.2
0.0
PctHisp



From scatterplot it can be concluded that there is are linear correlation between population and numbUrban.

Code Snippet:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix


CV=df.std()/data_frame.mean()
CV_= pd.DataFrame({'label':CV.index, 'CV':CV.values})
CV_=CV_.sort_values(by='CV', na_position='first', ascending=False)


top_features_array=np.array(CV_.values[:11, 1:2])
top_features=[]
```

```
for i in range(0,11):
    top_features.append(top_features_array[i][0])
plot_data=pd.DataFrame()
for i in range(0,11):
    plot_data[top_features[i]]=df[top_features[i]]

plot_data.plot(kind='box', subplots=True, layout=(4, 4), sharey=False, sharex=False)
plot_data.plot.box()
scatter_matrix(plot_data)
plt.show()
```

f:

Test Error of Linear Model = 0.0173255237864

Code Snippet:

```
import pandas as pd
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

df=pd.read_csv("C:/Users/Sarah Riaz/Documents/ML/HW/HW3/Crime-data/Crime-
data/communities.csv")
df=df.replace('?',0)

x= df.values[:1495, :126]
y= df.values[:1495, 126:]

model=linear_model.LinearRegression()
model.fit(x, y)

x_test= df.values[1495:, :126]
y_test= df.values[1495:, 126:]

predictions=model.predict(x_test)
error=mean_squared_error(y_test, predictions))
```

g:

Test Error of Ridge Regression = 0.016878582883

Lambda = 3.04303030303

Code Snippet:

```
x= df.values[:1495, :126]
y= df.values[:1495, 126:]

arr=np.linspace(0.01,100.1,100)
model=RidgeCV(alphas=arr,cv=5)
model.fit(x, y)


x_test= df.values[1495:, :126]
y_test= df.values[1495:, 126:]
predictions=model.predict(x_test)
error=mean_squared_error(y_test, predictions)
```

```
score = model.score(x_test, y_test)
lambda_ = model.alpha_
```

h:

Test Error of LASSO Model = 0.0170255841752

Score of LASSO Model = 0.641983459947

Lambda = 0.001

Selected variables:

state : -0.00102595958473

county : -0.000110547211674

community : -3.15417300321e-07

fold : -0.00233481420974

racepctblack : 0.207366166574

pctUrban : 0.0365518438209

pctWPubAsst : 0.0286071561541

AsianPerCap : 0.00332651469879

MalePctDivorce : 0.106438472374

PctKids2Par : -0.205817169956

PctYoungKids2Par : -0.0157406067529

PctWorkMom : -0.0500283974947

PctIlleg : 0.167396267735

PctRecImmig10 : 0.0197867930403

PctPersDenseHous : 0.136028424329

PctHousLess3BR : 0.0051148597567

HousVacant : 0.0850782967304

PctHousOccup : -0.0439604060465

PctVacantBoarded : 0.0516137106707

NumStreet : 0.0703805308156

PctForeignBorn : 0.00512017973444

PctSameCity85 : 0.00522171971593

PolicReqPerOffic : 0.00560692772946

PopDens : 0.00324871043047

LemasPctPolicOnPatr : 0.0152070886522

LemasGangUnitDeploy : 0.0197629428663

Test Error of LASSO with normalized features = 0.0197735263602

Score of LASSO with normalized features = 0.58419931914

Lambda = 0.001

Selected variables:

racePctWhite : -0.166575178423

PctKids2Par : -0.346807704431

PctIlleg : 0.164589579082

HousVacant : 0.0634186679487

Conclusion:

Test error is increased with normalized features because the number of selected variables is considerably decreased.

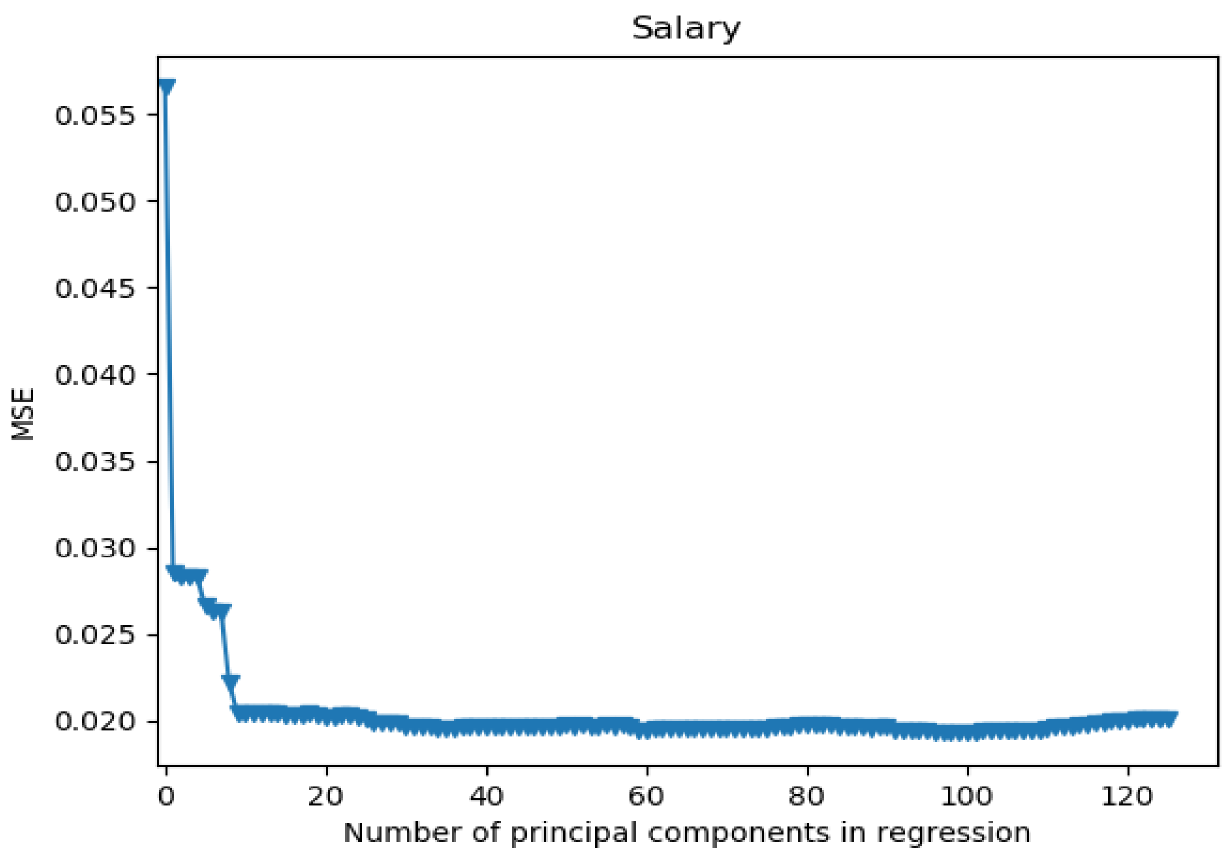
Code Snippet:

```
from sklearn.linear_model import LassoCV
x= data_frame.values[:1495, :126]
y= data_frame.values[:1495, 126:]

model=LassoCV(normalize=True,cv=5,alphas= np.linspace(0.001,100.1,3000))
#normalize=True for normalized features, otherwise False
model.fit(x, y)


x_test= df.values[1495:, :126]
y_test= df.values[1495:, 126:]

preds=model.predict(x_test)
test_error=mean_squared_error(y_test, preds)
```

```
score = model.score(x_test, y_test)

lambda_ = model.alpha_

final_features=model.coef_
print(type(final_features))
for i in range (0,126):
    if(final_features[i]!=0):
        selected_features = selected_features.append(final_features[i])
```

i:

PCR Model:



M (the number of principal components) = 98

Test Error = 0.0179137716061

Score = 0.62330652131

Code Snippet:

```
from sklearn.decomposition import PCA
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.preprocessing import scale
from sklearn.metrics import mean_squared_error

x= df.values[:1495, :126]
y= df.values[:1495, 126:]

pca = PCA()
X_reduced_train = pca.fit_transform(scale(x))
n = len(X_reduced_train)
# 10-fold CV with shuffle
kf_10 = model_selection.KFold( n_splits=10, shuffle=True, random_state=1)

mse = []
regr = LinearRegression()
score = -1*model_selection.cross_val_score(regr, np.ones((n,1)), y.ravel(), cv=kf_10,
scoring='neg_mean_squared_error').mean()
mse.append(score)
data={}
for i in np.arange(1, 126):
    score = -1*model_selection.cross_val_score(regr, X_reduced_train[:,:i], y.ravel(),
cv=kf_10, scoring='neg_mean_squared_error').mean()
    mse.append(score)
    data[i]=score

data=OrderedDict(sorted(data.items(), key=lambda t: t[1]))

plt.plot(np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Salary')
plt.xlim(xmin=-1)
plt.show()

M = list(data.keys())[0]

X_test=df.values[1495:,:126]
Y_test=df.values[1495:,126:]
X_reduced_test = pca.transform(scale(X_test))[:, :M + 1]

# Train regression model on training data
regr = LinearRegression()
regr.fit(X_reduced_train[:,:M+1], y)
predictions = regr.predict(X_reduced_test)
error_pcr=mean_squared_error(Y_test, predictions)
score = regr.score(X_reduced_test,Y_test)
```

i:

alpha for XGBoost L1 penalized gradient boosting tree = 7.895

Code Snippet:

```
import xgboost as xgb
from sklearn.grid_search import GridSearchCV
import pandas as pd
import numpy as np
x= df_train.values[:, 1:171]
y= df_train.values[:, :1]

optimized_GBM =
GridSearchCV(cv=5,estimator=xgb.XGBRegressor(),param_grid={'reg_alpha':
np.linspace(np.float_power(10, -4), np.float_power(10, 1), 20)},
        refit=True, scoring='neg_mean_squared_error', verbose=1)
```

```
optimized_GBM.fit(x, y)
scores = optimized_GBM.grid_scores_
```

Question 2:

b i:

Types of techniques used for dealing with data with missing values:

1) Deletion of these values
2) Imputation
    - Impute by Predictive Model(Decision Tree, Linear Regression)
    - Impute by Average (Replacing by mean or mode of the column)
3) Forward fill and backward fill

I have replaced missing values and changed the negative class to zero and positive class to one.

b ii:

Coefficient of variation of 170 features:

'cf_000' = 244.88836426184145
'co_000' = 244.51076535063208
'ad_000' = 244.37598592582142
'cs_009'237.93055371566217
'dh_000' 123.21609721755667
'dj_000' 117.49422514513171
'ag_000' 92.91775503608642
'as_000' 87.33249956499247
'ay_009' 84.73373459937712
'ak_000' 80.42497540906561
'az_009' 77.83854428850402
'ch_000' 77.4538571344374
'au_000' 68.88275094860896
'cr_000' 58.07807152884354]
'ay_001' 52.82471400357465
'df_000' 52.292813588128965
'dz_000' 51.3322277688843
'ef_000' 49.36665925628379
'cs_008' 48.220079107190436
'aj_000' 44.26599598905513
'eg_000' 42.48174746009007
'dl_000' 39.73908782176686
'ay_002' 39.24865364892385
'dg_000' 38.48616191142954
'ay_000' 37.42828471068997
'dk_000' 37.03912307273606
'cy_000' 36.60090813287275

'dm_000' 36.26140326564316
'ag_001' 35.24931353569572
'ea_000' 34.94651895491634
'cn_009' 34.27327078908045
'ay_004' 33.75234540476702
'ag_009' 33.35756746889936
'da_000' 29.367526519061453
'ay_003' 28.735090223602636
'cn_000' 26.335574345711866
'ae_000' 24.200136597481663
'at_000' 23.708186710574548
'az_008' 22.679650237535288
'dq_000' 22.030103070594105
'af_000' 19.4712950562839
'ai_000' 18.203806264011888
'ag_002' 17.56590713284474
'az_007' 16.229426743966997
'cl_000' 14.970729682408079
'cz_000' 14.55100707851276
'cp_000' 13.949635339265187
'az_002' 13.290748537881425
'ay_005' 12.524654611577459
'di_000' 11.826232152583813
'ar_000' 11.35434651281637
'cn_001' 11.26256085031939
'cj_000' 11.069531465621136
'ab_000' 10.383493866617679
'cn_008' 9.744570095082587
'az_000' 9.434445752852419
'ba_009' 9.430241407037293
'al_000' 9.173106315332872
'am_0' 9.155221404503994
'az_006' 8.880859131119804
'ag_003' 8.647402475930393
'ct_000' 8.516292469155456
'dy_000' 7.796648367537321
'az_001' 7.733630366859148
'classs' 7.681209758219827
'az_003' 7.530939390242064
'bf_000' 7.462350803533416
'bc_000' 7.274336193370247
'cu_000' 7.134734818543998
'be_000' 6.887037454948392
'dr_000' 6.8654458053286405

'ba_008' 6.830710218280034
'cn_002' 6.70740290737369
'cn_007' 6.346280207398323
'bz_000' 6.316860672188324
'db_000' 6.259495274867568
'ag_008' 6.225098758984985
'av_000' 6.033640551734054
'ee_009' 5.691612313066102
'ag_004' 5.463603548437661
'cs_007' 5.450834018431791
'cm_000' 5.4180729882698895
'dx_000' 5.374132379126713
'bd_000' 5.371019480888662
'cs_002' 5.119683093431874
'ee_007' 5.019734094778544
'cx_000' 4.935251999594238
'cg_000' 4.717482545409535
'cs_004' 4.696075538143602
'de_000' 4.613376473854331
'eb_000' 4.561663794029881
'cn_003' 4.195532267195919
'ax_000' 4.051600091961907
'ay_008' 3.8198300949707136
'cs_001' 3.68299249714019
'dv_000' 3.5968313156621043
'bj_000' 3.5872346650103997
'ay_007' 3.323088393431945
'ee_000' 3.314736519414211
'ee_001' 3.298913993651963
'ee_008' 3.260185601566189
'ee_006' 3.22998123654064
'cn_006' 3.1722971106568476
'cs_003' 3.1455577252377824
'dd_000' 3.1189351223710817
'ap_000' 3.0939997750845376
'ck_000' 3.0623165631663043
'ay_006' 3.059127327981951
'ba_006' 3.0440620186488476
'az_005' 3.043953611397488
'bi_000' 3.0339490484047236
'br_000' 3.0269475459043695
'bq_000' 2.971601774161997
'ag_005' 2.9621066820642303
'du_000' 2.925292934460321

'ba_002' 2.9141392916849345
'ec_00' 2.9056200333435585
'dn_000' 2.903214785061223
'bp_000' 2.8705151562076527
'aq_000' 2.869491603335408
'ag_007' 2.867503049354326
'ee_005' 2.8637008957828605
'az_004' 2.8509856819000663
'ba_007' 2.8450073344514903
'ba_003' 2.749732187640745
'bo_000' 2.7411691910958322
'ba_000' 2.7163353472921985
'ba_005' 2.713100046440876
'ed_000' 2.6828506487544375
'ba_004' 2.6485568572349028
'bh_000' 2.6430907143606017
'ba_001' 2.6411890158096427
'ee_004' 2.6380240917274773
'cn_004' 2.63436991391018
'cc_000' 2.6133481412656425
'ee_002' 2.6106577147106385
'bx_000' 2.5995630658141926
'ee_003' 2.590124055687687
'bn_000' 2.552650132444224
'cs_005' 2.527568168504519
'by_000' 2.462328281127565
'bt_000' 2.451896061495529
'aa_000' 2.450937577943998
'bu_000' 2.421498334642216
'bv_000' 2.4214981698459077
'cq_000' 2.4214981359196712
'bb_000' 2.420562031330786
'ci_000' 2.4082840520729047
'ds_000' 2.386890809599732
'ag_006' 2.3738311833499317
'cn_005' 2.3609029920656384
'ah_000' 2.327518597655898
'bg_000' 2.3250156589461577
'ac_000' 2.310240723932622
'ao_000' 2.2847268999382497
'ce_000' 2.2774271782222115
'an_000' 2.2653994478781003
'dt_000' 2.2566021209926292
'bm_000' 2.2312129246168433

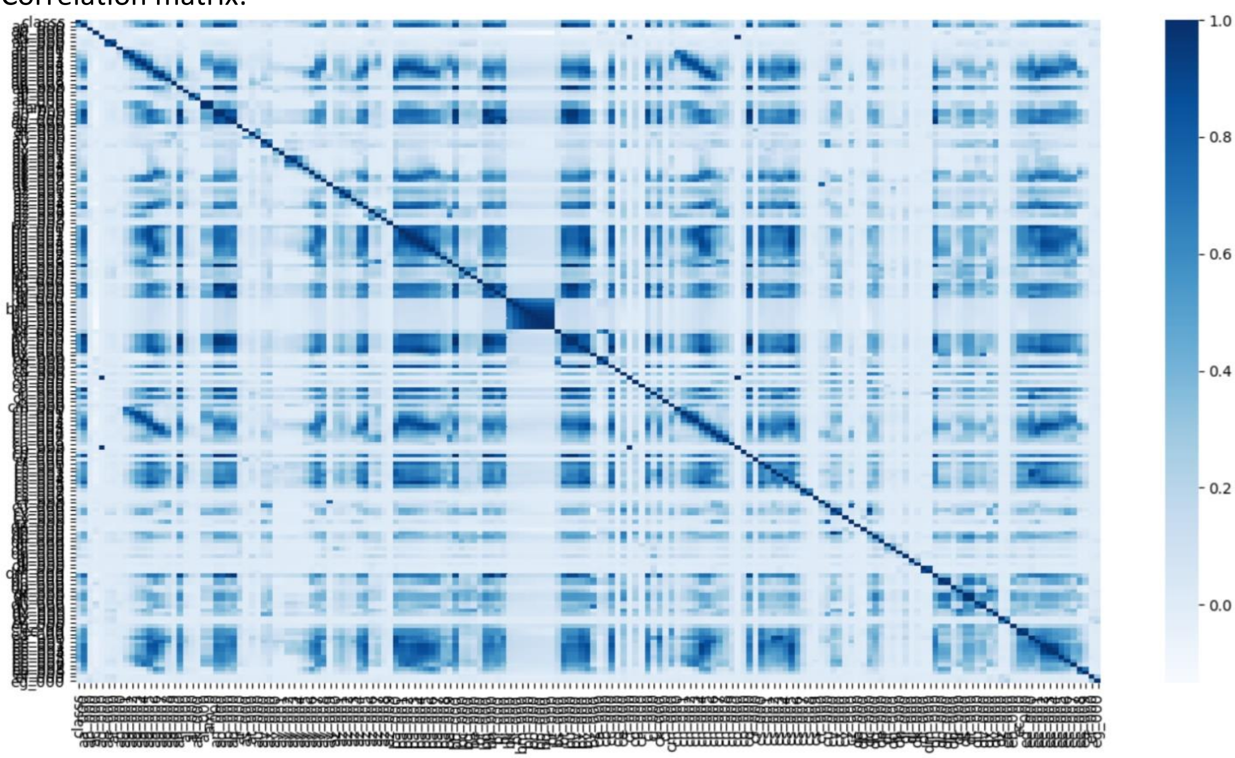
'cv_000' 2.2311750209151135
'do_000' 2.2099856627107117
'dc_000' 2.197618552624876
'cs_006' 2.1550953733897757
'dp_000' 2.0642138575664157
'cs_000' 1.8957196403512118
'bl_000' 1.625658055074731
'bk_000' 1.4256062927273956
'bs_000' 1.0638627687031057
'ca_000' 1.0136149669591161
'cb_000' 0.9228512609832727
'cd_000' 0.10674849332694764

Code Snippet:

```
import numpy as np
import pandas as pd

CV=df_train.std()/df_train.mean()
CV_= pd.DataFrame({'label':CV.index, 'CV':CV.values})
CV_=CV_.sort_values(by='CV', ascending=False, na_position='first')
```

b iii:

Correlation matrix:



Code Snippet:

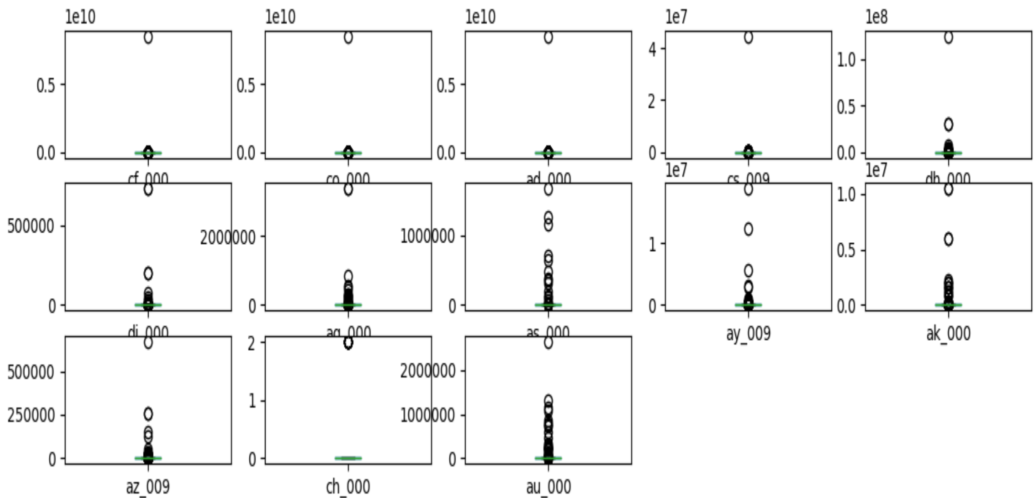
```
corr=df_train.corr()

plt.figure(figsize=(100,100))
```

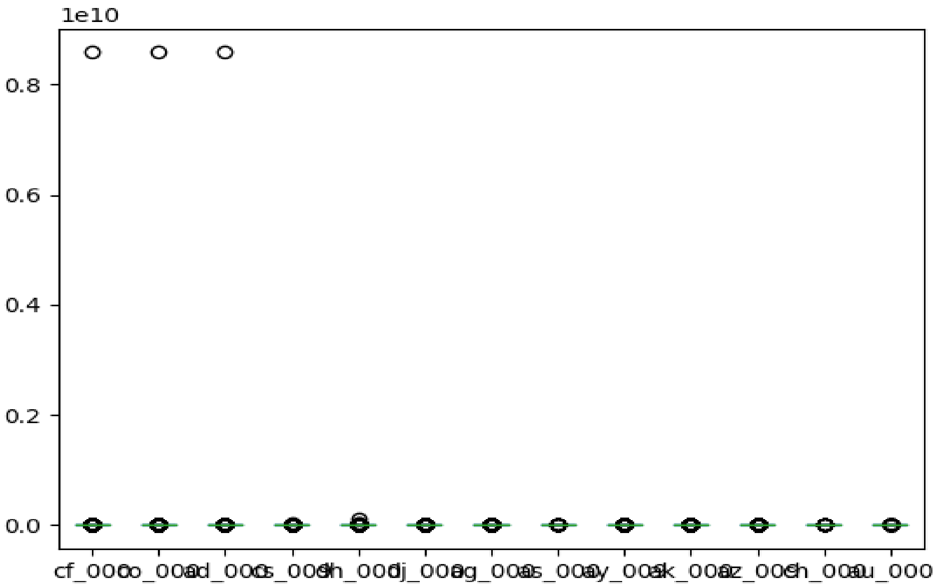
```
sns.heatmap(corr, cmap="Blues")
plt.show()
```

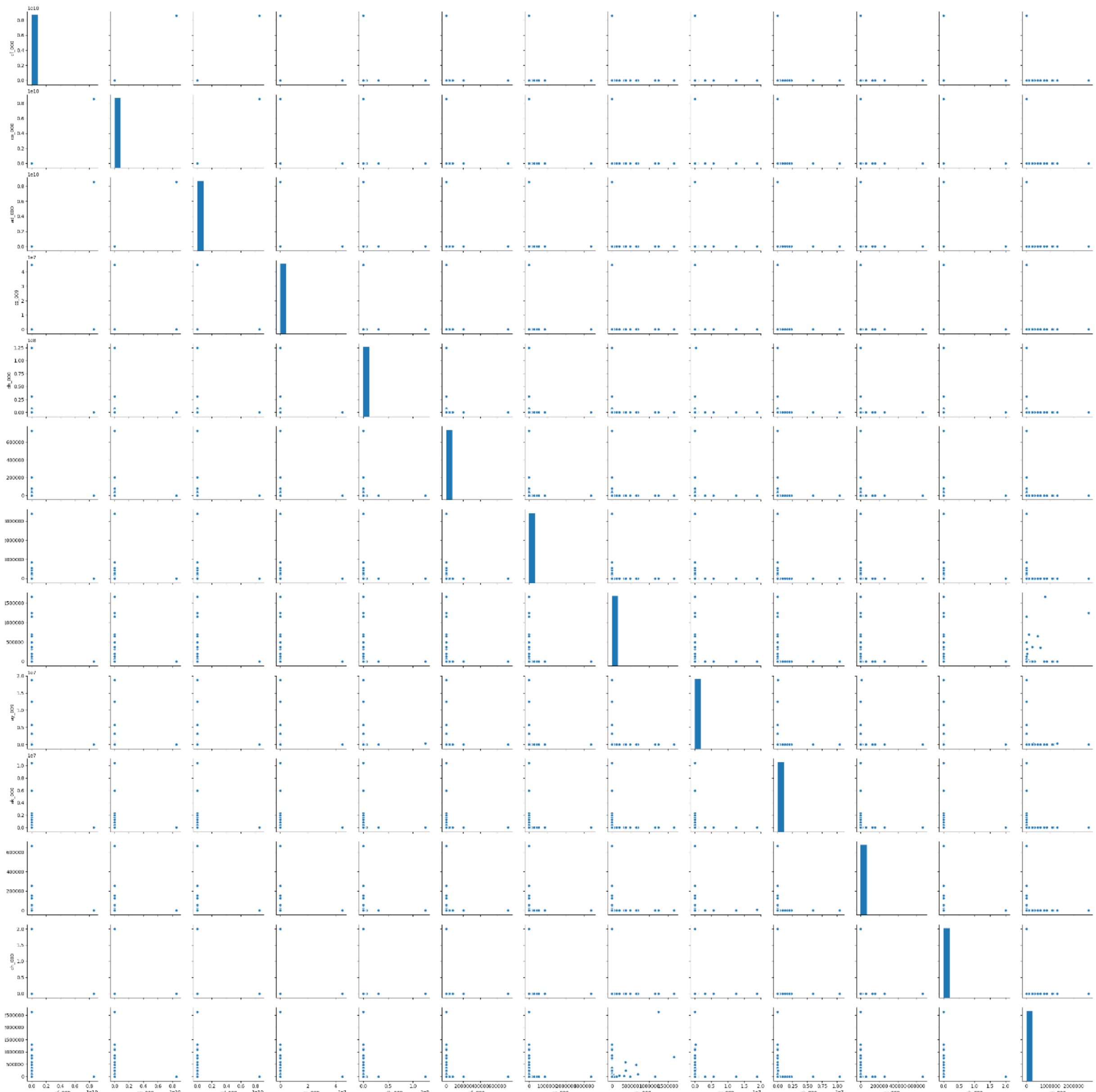
b iv:

Top √170 features:

cf_000, co_000, cs_009, ch_000, dh_000, dj_000, ad_000, ag_000, as_000, ay_009, ak_000 , au_000 and az_009




1e10
0.8
0.6
0.4
0.2
0.0
au_000



Scatter-matrix are hard to conclude from, therefore we need to look at the coefficients to draw conclusion about significance of these features.

Code Snippet:

```
CV=df.std()/df.mean()
CV= pd.DataFrame({'label':CV.index, 'CV':CV.values}).sort_values(by='CV',
ascending=False, na_position='first')

top_features=[]

for i in range(0,13):
    top_features.append(np.array(CV.values[:13,1:2])[i][0])

plot_data=pd.DataFrame()
for i in range(0,13):
```

```
    plot_data[top_features[i]]=df[top_features[i]]
plt.figure()
plot_data.plot(kind='box', subplots=True, layout=(5, 5), sharex=False, sharey=False)
plot_data.plot.box()
scatter_matrix(plot_data)
plt.show()
plt.savefig()
```

b v:

Train set negative data = 59000

Train set positive data = 1000

Test set negative data = 15625

Test set positive data = 375

Code Snippet:

```
df_majority_train = df_train[df_train.classs == "neg"]
df_minority_train = df_train[df_train.classs == "pos"]

df_majority_test = df_test[df_test.classs == "neg"]
df_minority_test = df_test[df_test.classs == "pos"]
```

c:

Random Forest Classifier with imbalance train error:

missclassification rate = 0.012033

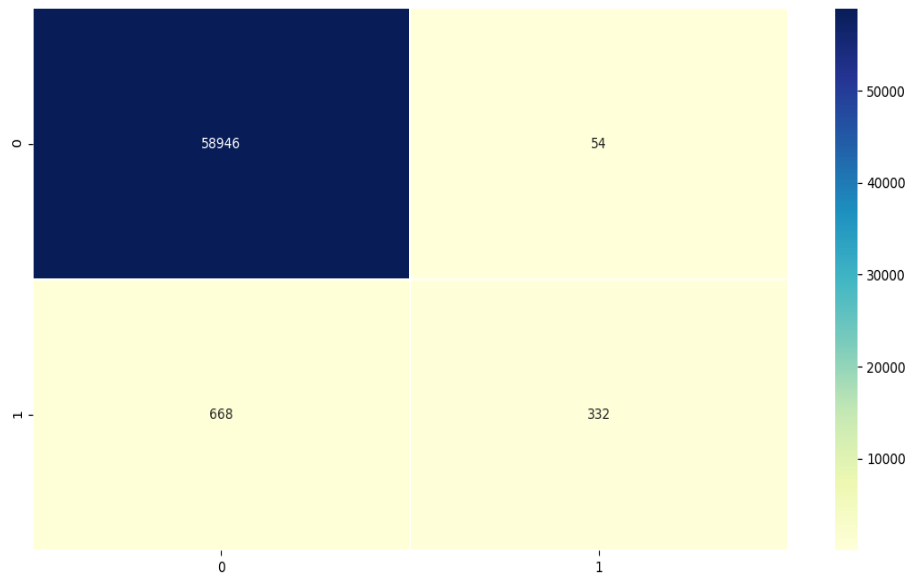
OOB error = 0.012900000000000023

Random Forest Classifier with imbalance test Error:

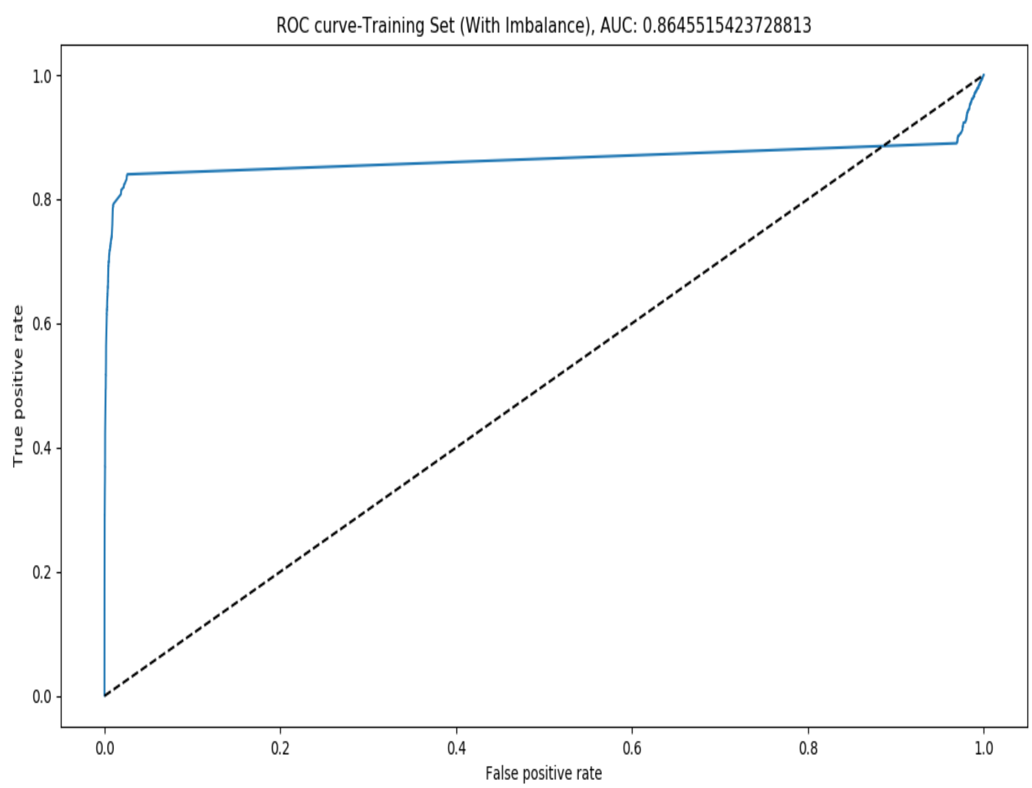
missclassification rate = 0.016

OOB error = 0.01254999999995

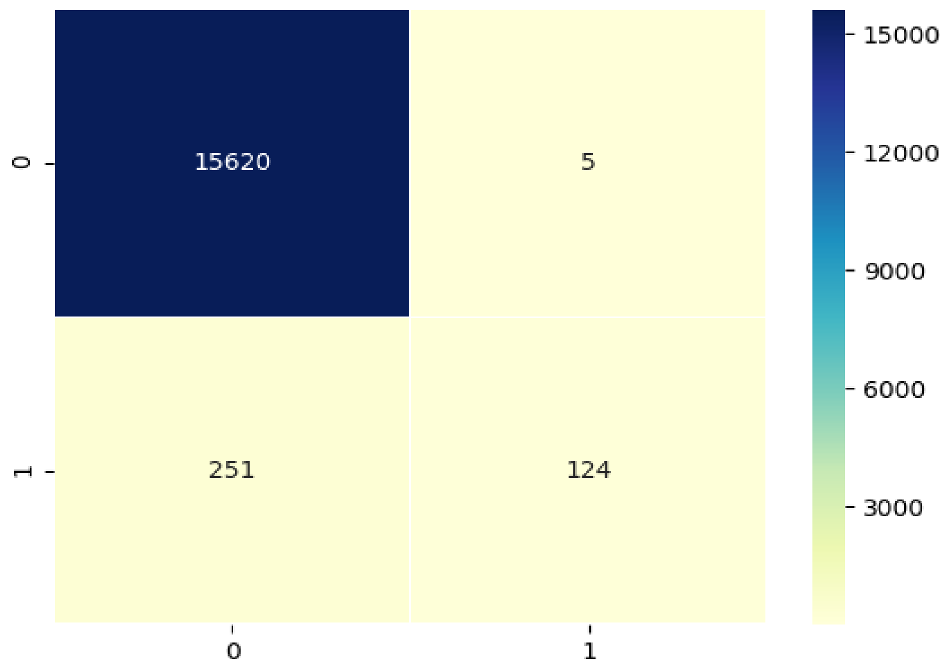
OOB error is better than the misclassification error in both training and testing.

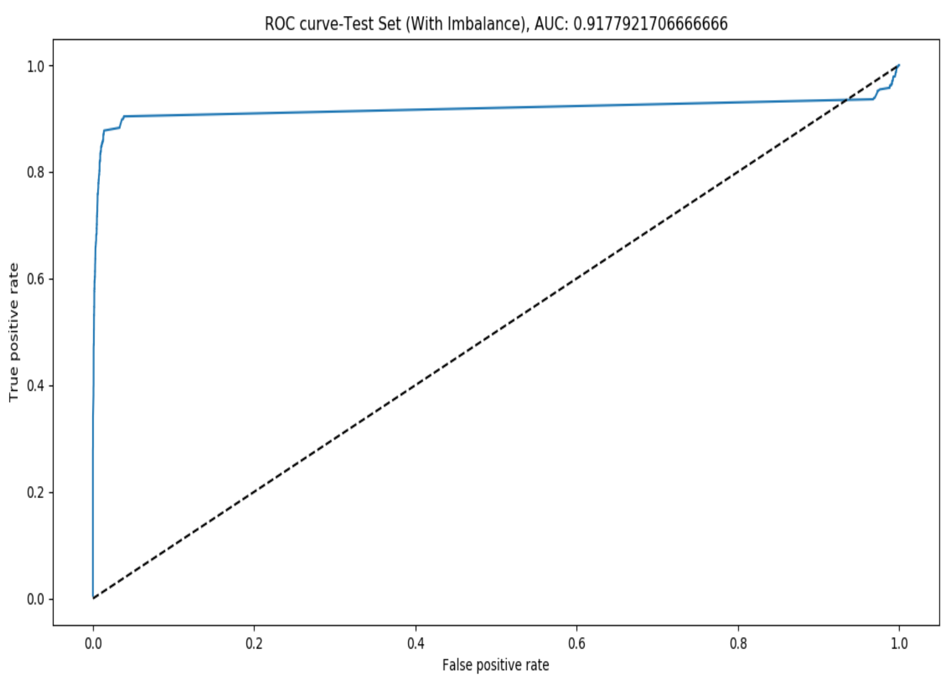


58946
54
668
332
0
1
0
1
50000
40000
30000
20000
10000



ROC curve-Training Set (With Imbalance), AUC: 0.8645515423728813
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0





Code Snippet:

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import LinearSVC,SVC
```

```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve,auc

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

X_train=df_train.values[:,1:171]
Y_train=df_train.values[:,:1]


X_test=df_test.values[:,1:171]
Y_test=df_test.values[:,:1]

model=RandomForestClassifier(max_depth=5)
model.fit(X_train,Y_train.ravel())

rf = RandomForestClassifier(max_depth=3,oob_score=True)
rf_enc = OneHotEncoder()
rf_lm = SVC(probability=True)

rf.fit(X_train, Y_train.ravel())
rf_enc.fit(rf.apply(X_train))
model_used=rf_lm.fit(rf_enc.transform(rf.apply(X_test)), Y_test.ravel())
preds=rf.predict(X_test)


y_pred_rf_lm = rf_lm.predict_proba(rf_enc.transform(rf.apply(X_test)))[:, 1]
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(Y_test, y_pred_rf_lm,pos_label='pos')
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)

plt.plot(fpr_rf_lm, tpr_rf_lm, label=str(roc_auc))
plt.title('ROC curve-Test Set (With Imbalance), AUC: '+str(roc_auc))
plt.xlabel('False positive rate')
plt.plot([0, 1], [0, 1], 'k--')
plt.ylabel('True positive rate')
plt.show()

confusion_matrx=confusion_matrix(Y_test,preds)
sns.heatmap(confusion_matrx,cmap="YlGnBu",annot=True,linewidths=.5,fmt='d')
plt.show()

missclassifications=0
for i in range(0,2):
    for j in range(0,2):
        if i!=j:
            missclassifications+=confusion_matrx[i][j]
```

d:

Resampling, downsampling and upsampling are used to remove imbalance.
Random Forest Classifier without imbalance train error:

missclassification rate = 0.049813559322

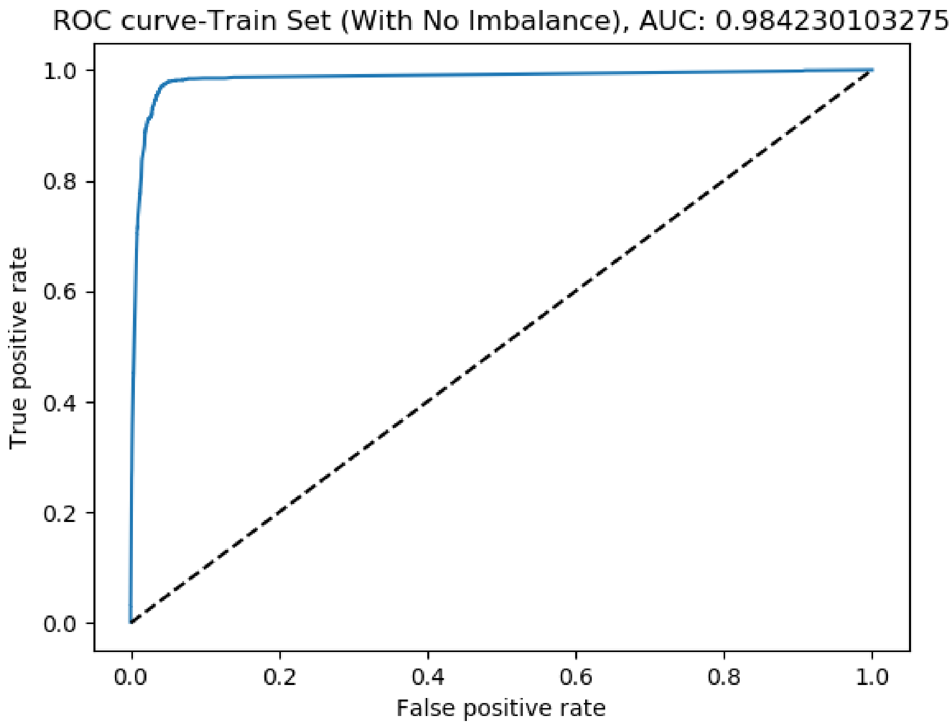
OOB error = 0.0548728813559

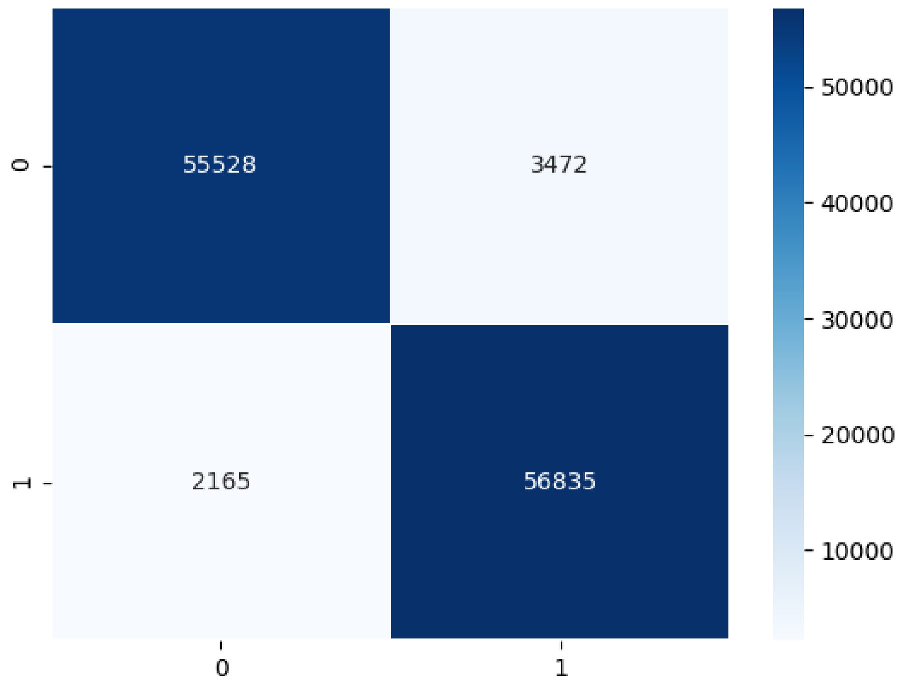
Random Forest Classifier without imbalance test error:

missclassification rate = 0.042016

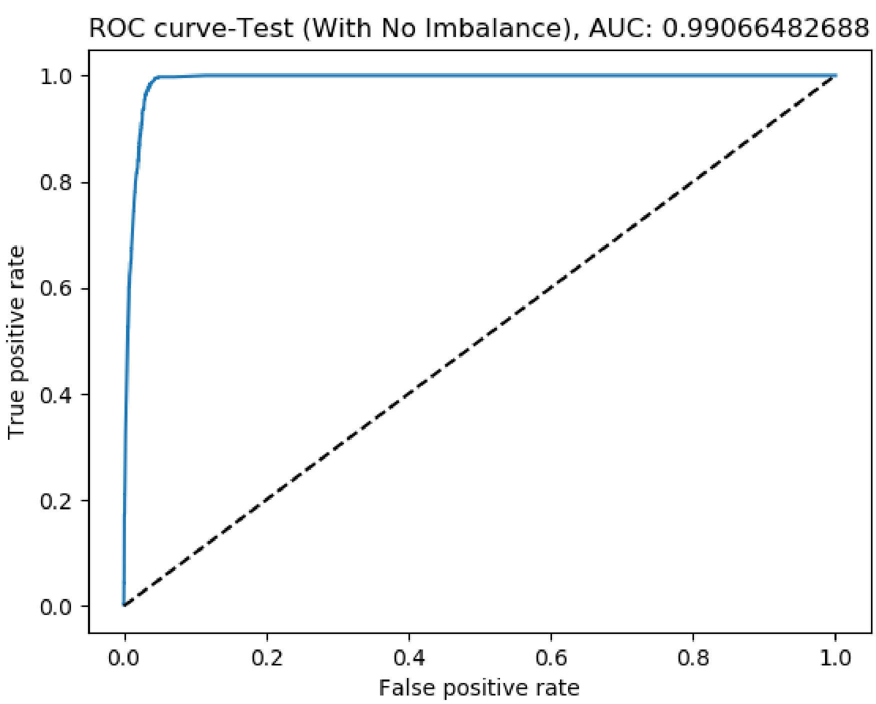
OOB Error = 0.0567627118644

Error has significantly reduced because of resampling, therefore it improves the model.

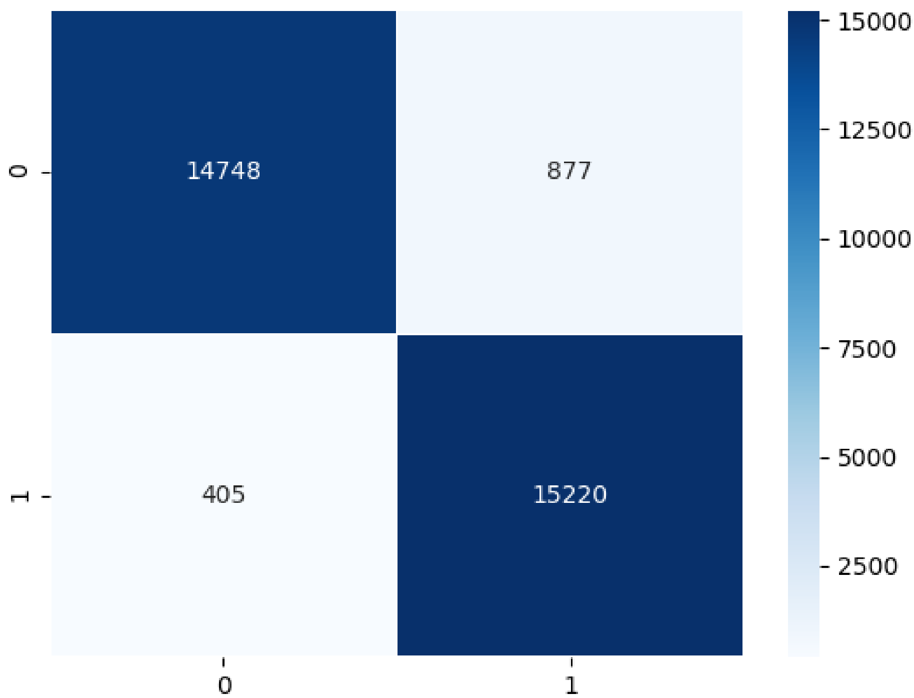




55528
3472
2165
56835
0
1
0
1
50000
40000
30000
20000
10000



ROC curve-Test (With No Imbalance), AUC: 0.99066482688
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0



Code Snippet:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import log_loss

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve,auc

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.utils import resample

df_minority_upsampled = resample(df_minority, replace=True, n_samples=59000,
random_state=123)

df_upsampled = pd.concat([df_majority, df_minority_upsampled])

df_majority_test = df_test[df_test.classs == "neg"]
df_minority_test = df_test[df_test.classs == "pos"]

df_minority_upsampled_test = resample(df_minority_test, replace=True, n_samples=15625,
random_state=123

df_upsampled_test = pd.concat([df_majority_test, df_minority_upsampled_test])

df_upsampled.replace('na',0,inplace=True)
df_upsampled_test.replace('na',0,inplace=True)

X_train=df_upsampled.values[:,1:171]
```

```
Y_train=df_upsampled.values[:,:1]
model=RandomForestClassifier(max_depth=5)
model.fit(X_train,Y_train.ravel())

model_prob = model.predict_proba(X_train)
score=log_loss(Y_train,model_prob)

model_prob=model_prob.reshape(1,-1)
rf = RandomForestClassifier(max_depth=3,oob_score=True)
rf_enc = OneHotEncoder()
rf_lm = LogisticRegression()
rf.fit(X_train, Y_train.ravel())
rf_enc.fit(rf.apply(X_train))
model_used=rf_lm.fit(rf_enc.transform(rf.apply(X_train)), Y_train.ravel())
preds=rf.predict(X_train)

y_pred_rf_lm = rf_lm.predict_proba(rf_enc.transform(rf.apply(X_train)))[:, 1]
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(Y_train, y_pred_rf_lm,pos_label='pos')
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)

plt.plot(fpr_rf_lm, tpr_rf_lm, label=str(roc_auc))
plt.title('ROC curve-Train Set (With No Imbalance), AUC: '+str(roc_auc))
plt.xlabel('False positive rate')
plt.plot([0, 1], [0, 1], 'k--')
plt.ylabel('True positive rate')
plt.show()
confusion_matrx=confusion_matrix(Y_train,preds)
sns.heatmap(confusion_matrx,cmap="Blues",annot=True,linewidths=.5,fmt='d')
plt.show()
missclassifications=0
for i in range(0,2):
    for j in range(0,2):
        if i!=j:
            missclassifications+=confusion_matrx[i][j]
```

e:

Model Trees with weka classifier (LMT) training err:

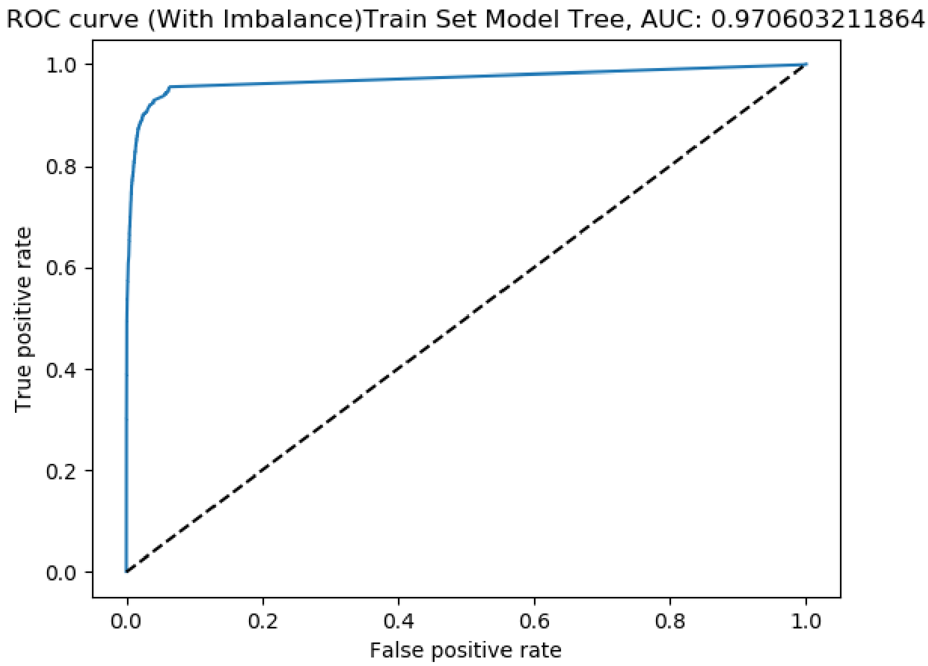
missclassification rate = 0.0117666666667

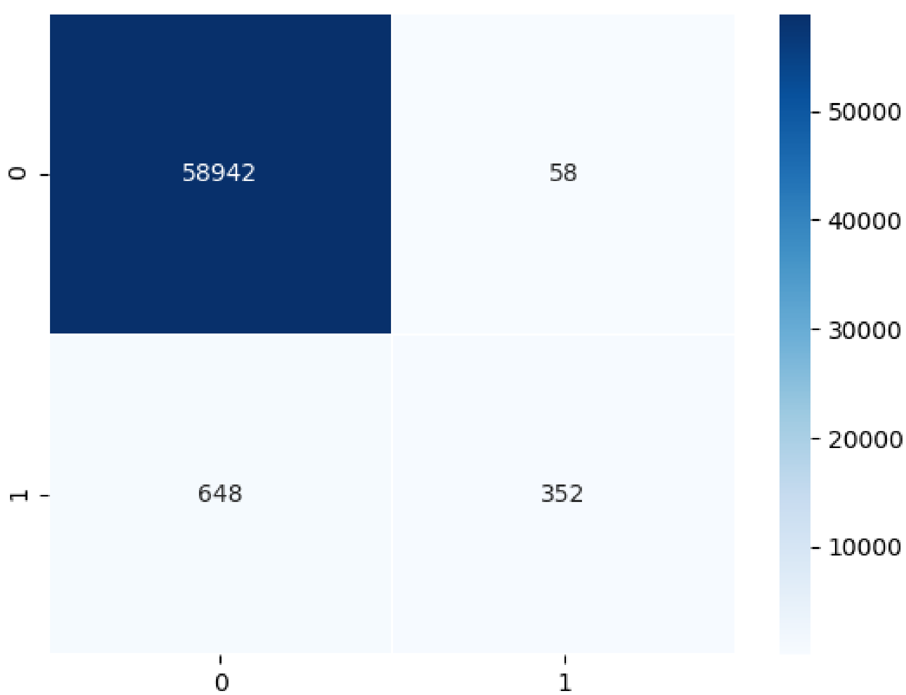
OOB error = 0.0129

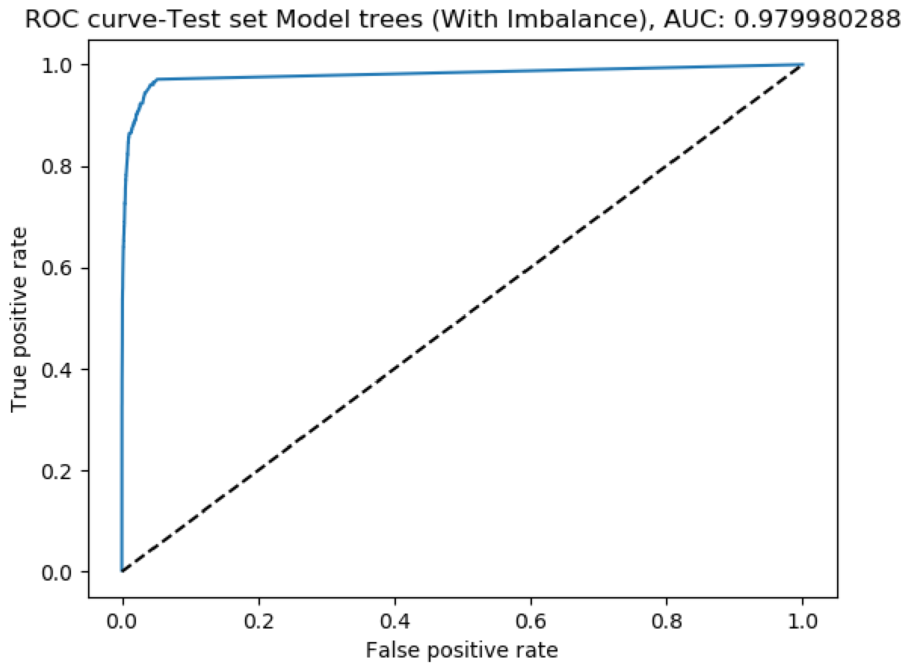
Model Trees with weka classifier (LMT) testing err:

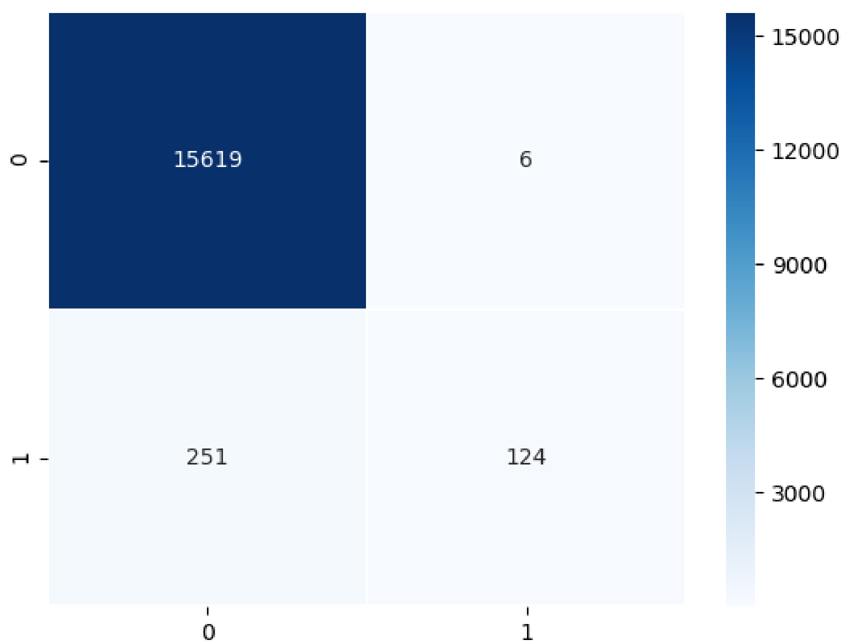
missclassification rate = 0.0160625

OOB error = 0.01245


ROC curve (With Imbalance)Train Set Model Tree, AUC: 0.970603211864
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0


58942
58
648
352
0
1
50000
40000
30000
20000
10000





Code Snippet:

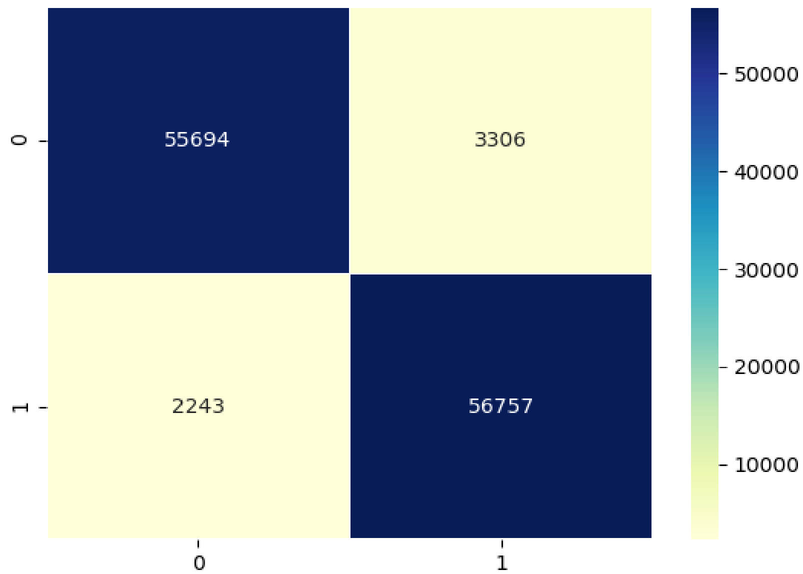
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import OneHotEncoder
```

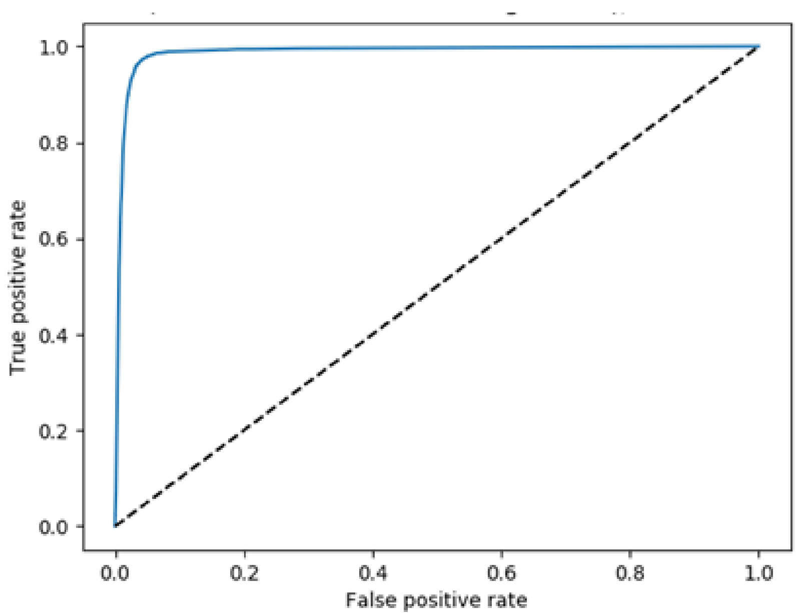
```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve,auc

from sklearn.ensemble import RandomForestClassifier

df_train.replace('na',0,inplace=True)
df_test.replace('na',0,inplace=True)

X_train=df_train.values[:,1:171]
Y_train=df_train.values[:,:1]


rf = RandomForestClassifier(max_depth=3,oob_score=True)
rf_enc = OneHotEncoder()
rf_lm = LogisticRegressionCV(cv=5)

rf.fit(X_train, Y_train.ravel())
rf_enc.fit(rf.apply(X_train))
model_used=rf_lm.fit(rf_enc.transform(rf.apply(X_train)), Y_train.ravel())
preds=rf.predict(X_train)


y_pred_rf_lm = rf_lm.predict_proba(rf_enc.transform(rf.apply(X_train)))[:, 1]
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(Y_train, y_pred_rf_lm,pos_label='pos')
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)

plt.plot(fpr_rf_lm, tpr_rf_lm, label=str(roc_auc))
plt.title('ROC curve (With Imbalance)Train Set Model Tree, AUC: '+str(roc_auc))
plt.xlabel('False positive rate')
plt.plot([0, 1], [0, 1], 'k--')
plt.ylabel('True positive rate')
plt.show()

confusion_matrx=confusion_matrix(Y_train,preds)
sns.heatmap(confusion_matrx,cmap="Blues",annot=True,linewidths=.5,fmt='d')
plt.show()

missclassifications=0
for i in range(0,2):
    for j in range(0,2):
        if i!=j:
            missclassifications+=confusion_matrx[i][j]
```

f:

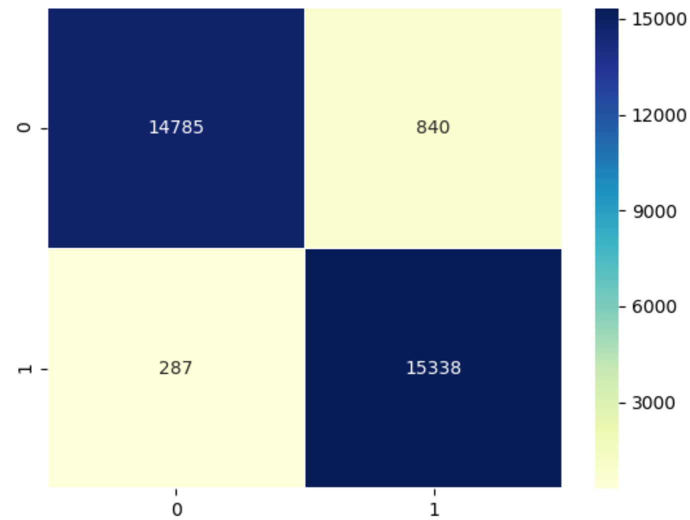
Model trees with SMOTE Filter training misclassification rate = 0.04755084745762712

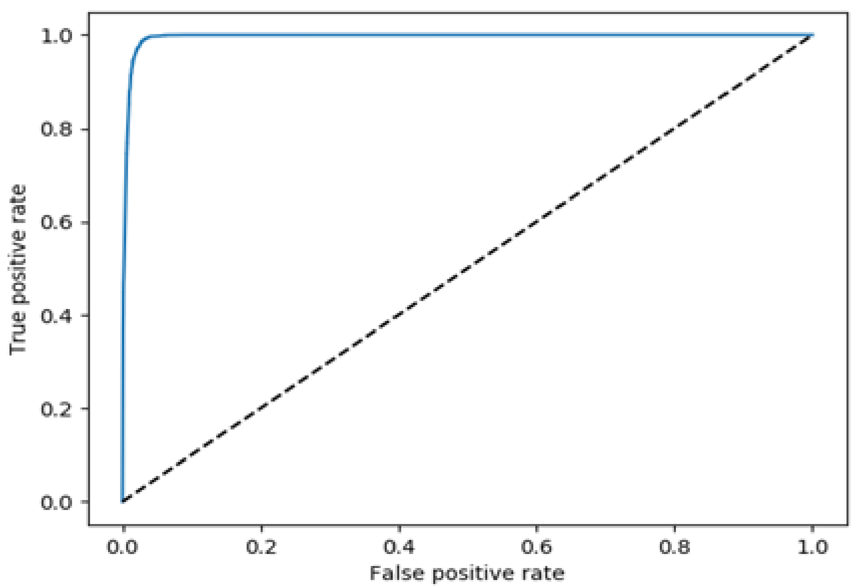
AUC = 0.9875381213





Model trees with SMOTE Filter testing misclassification rate = 0.036064
AUC = 0.995319769





False positive rate

Code Snippet:

```
import weka.core.jvm as jvm
from weka.classifiers import Classifier
from weka.flow.control import Flow, Branch, Sequence
from weka.classifiers import FilteredClassifier
from weka.core.converters import Loader
from weka.classifiers import Evaluation
from weka.core.classes import Random
from weka.filters import Filter
import seaborn as sns
import matplotlib as plt

import weka.plot.classifiers as plcls

jvm.start()
```

```
loader = Loader(classname="weka.core.converters.CSVLoader")
data = loader.load_file("/aps.failure_training_set.csv")

remove = Filter(classname="weka.filters.supervised.instance.SMOTE", options=["-R", "1-
3"])

cls = Classifier(classname="weka.classifiers.trees.LMT")

fc = FilteredClassifier()
fc.classifier = cls

evl = Evaluation(data)
evl.crossvalidate_model(fc, data, 10, Random(1))
conf=evl.confusion_matrix
sns.heatmap(conf,cmap="YlGnBu",annot=True,linewidths=.5,fmt='d')

plcls.plot_roc(evl, class_index=[0, 1], wait=True)
plt.show()
```

DATE........../........../.........   SUBJECT:

Question 5

ISLR - 6.8.3:

a) With increase in s, there is less constraint on β_j, therefore model becomes flexible and training RSS decreases (TRUE)

b) With increase in s, β_j has less strict contraint, therefore model becomes more flexible and RSS decreases initially but after a certain point it starts increasing, thus making a U-shape. (TRUE)

c) There is a steady increase in variance because as s increases, the model becomes more flexible. (TRUE)

d) Bias always decreases with more flexible model. (TRUE)

e) The irreducible error is not related to model selection because it's a constant value. (TRUE)

Question 4

ISLR-6.8.5:

a) $x_{11} = x_{12} = x_1$, $x_{21} = x_{22} = x_2$

In Ridge regression, we try to minimize:

$(y_1 - \hat\beta_1 x_1 - \hat\beta_2 x_1)^2 + (y_2 - \hat\beta_1 x_2 - \hat\beta_2 x_2) + \lambda(\hat\beta_1^2 + \hat\beta_2^2)$

b) Differentiating $\hat\beta_1, \hat\beta_2$ and equating to zero:

$\hat\beta_1(x_1^2 + x_2^2 + \lambda) + \hat\beta_2(x_1^2 + x_2^2) = y_1 x_1 + y_2 x_2 \rightarrow A$

$\hat\beta_1(x_1^2 + x_2^2) + \hat\beta_2(x_1^2 + x_2^2 + \lambda) = y_1 x_1 + y_2 x_2 \rightarrow B$

Subtracting B from A:

$\hat\beta_1 \lambda = \hat\beta_2 \lambda \Rightarrow \hat\beta_1 = \hat\beta_2$

c) Lasso:

$((y_1 - \hat\beta_1 x_1) - \hat\beta_2 x_1)^2 + (y_2 - \hat\beta_1 x_2 - \hat\beta_2 x_2)^2 + \lambda(|\hat\beta_1| + |\hat\beta_2|)$

(we want to minimize this)

d)

$(y_1 - \hat\beta_1 x_1 - \hat\beta_2 x_1)^2 + (y_2 - \hat\beta_1 x_2 - \hat\beta_2 x_2)^2$

given: $\sum_{i=1}^{n} |\hat\beta_i| \leq s$

Lasso constraint has a diamond shape with center at origin of $(\hat\beta_1, \hat\beta_2)$.

if: $x_{11} = x_{12} = x_1$, $x_{21} = x_{22} = x_2$, $x_1 + x_2 = 0$, $y_1 + y_2 = 0$

We want to minimize:

$2\{y_1 - (\hat\beta_1 + \hat\beta_2 x_1)\}^2 \geq 0$

d) Replacing the constraint term in part(b), the derivative term to β is:

$$\frac{\partial}{\partial \hat{\beta}} (\lambda|\beta|) : \lambda \frac{|\beta|}{\beta}$$

Following the steps in (b), we get:

$$\lambda \frac{|\beta_1|}{\beta_1} = \lambda \frac{|\beta_2|}{\beta_2}$$

So, the lasso just requires β_1 & β_2 both positive or negative (ignoring 0).

Question: 5.

ISLR: 8.4.5

Using majority polling, X will be classified as Red because there are six red and four green classifications in the given data.

Using average probability, ~~we~~ X will bee classified as green because average of 10 probabilities is 0.45.

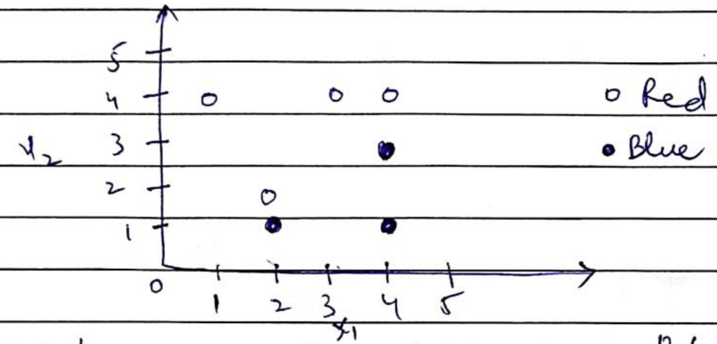
Question 6:

ISLR 9.7.3.

a) $x_1 = c(3, 2, 4, 1, 2, 4, 4)$

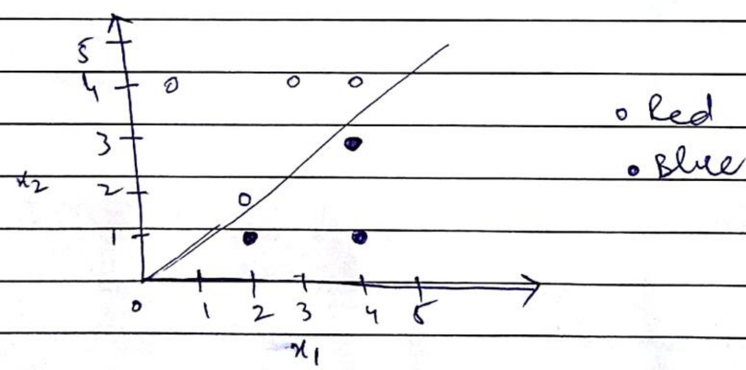
$x_2 = c(4, 2, 4, 4, 1, 3, 1)$

colors = c(red, red, red, red, blue, blue, blue)



plot (x_1, x_2, col = colors, xlim = c(0,5), ylim(0,5)).

b)



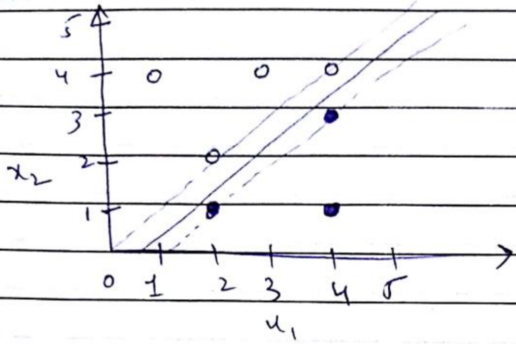
c) If $x_1 - x_2 - 0.5 < 0$

classify red.

else:

classify blue.

d)



e) Support vectors = (2,1), (2,2), (4,3), (4,4)

f) If we move (4,1) or (3,4), (1,4), maximal hyperplane will not change because they are not support vectors.

g) $x_1 - x_2 - 0.3 = 0$ (not an optimal hyperplane)

h)