ed_squad_final_smn

July 21, 2021

1 DS5110 Final Project Assignment

1.1 The Ed Squad

Shilpa Narayan (smn7ba) Ashlie Ossege (ajo5fs) Jamie Oh (hso6b) Isaac Stevens (is3sb)

1.1.1 About data

AMERICAN COMMUNITY SURVEY 2015-2019 5-YEAR SAMPLE 5-in-100 national random sample of the population Contains all households and persons from the 1% ACS samples for 2015, 2016, 2017, 2018, and 2019 identifiable by year. The data include persons in group quarters. This is a weighted sample. The smallest identifiable geographic unit is the PUMA, containing at least 100,000 persons. PUMAs do not cross state boundaries.

The lowest unit of geography in the microdata files is still the PUMA. PUMAs contain at least 100,000 people. Aggregate data (but not microdata) is currently available from the Census Bureau for geographic areas as small as block groups, but only for the entire 2005-2009 period.

PERNUM numbers all persons within each household consecutively in the order in which they appear on the original census or survey form. When combined with SAMPLE and SERIAL, PERNUM uniquely identifies each person within the IPUMS.

MULTYEAR identifies the actual year of survey in multi-year ACS/PRCS samples.

For example, the 3-year ACS and PRCS data files each include cases from three single-year files. For these multi-year samples, the YEAR variable identifies the last year of data (2007 for the 2005-2007 3-year data; 2008 for the 2006-2008 data; and so on). MULTYEAR gives the single-year sample from which the case was drawn (2005, 2006, or 2007 for the 2005-2007 3-year data; 2006, 2007, or 2008 for the 2006-2008 3-year data; and so on).

https://usa.ipums.org/usa/acs_multyr.shtml

[1]:
```
#import spark packages
from pyspark.sql import SparkSession
from pyspark.sql.types import ArrayType, StructField, StructType, StringType,␣
↪IntegerType
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import Correlation
from pyspark.sql import functions as F
from pyspark.sql.types import *
from pyspark.sql import SQLContext
```

```
#import mlLib libraries for classification
from pyspark.sql.functions import col
from pyspark.ml.tuning import CrossValidator,␣
↪ParamGridBuilder,TrainValidationSplit
from pyspark.ml.evaluation import␣
↪BinaryClassificationEvaluator,MulticlassClassificationEvaluator
from pyspark.ml.classification import RandomForestClassifier, LinearSVC,␣
↪LogisticRegression, GBTClassifier
from pyspark.ml import Pipeline
from pyspark.ml.feature import PCA, Binarizer
from pyspark.mllib.evaluation import␣
↪MulticlassMetrics,BinaryClassificationMetrics
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
```

[2]:

```
#import python packages too for visualizations
%matplotlib inline
import plotly.express as px
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import itertools
from sklearn.metrics import confusion_matrix
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_rows', 200000)
sns.set(rc={'figure.figsize':(10,8)})
sns.set_style("white")
```

[3]:

```
#set seed so results are reproducible
seed = 42
```

[4]:

```
# set up the spark session
spark = SparkSession \
    .builder \
    .master("local") \
    .appName("Ed Squad Project") \
    .config("spark.executor.memory", '200g') \
    .config('spark.executor.cores', '8') \
    .config('spark.cores.max', '8')\
    .config("spark.driver.memory", '200g')\
    .getOrCreate()

sc = spark.sparkContext
sqlContext = SQLContext(spark)
```

[5]:
```
sc.uiWebUrl
```

[5]: 'http://udc-aw34-8c1:4040'

1.2 Read In Data

[7]:
```
#import acs sample data for 2015-2019 and south region
data = spark.read.csv('/project/ds5559/ds5110_project_snoo/acs_15_19_south_puma.
↪csv', inferSchema="true", header="true")
```

```
CPU times: user 13.3 ms, sys: 1.3 ms, total: 14.6 ms
Wall time: 1min 16s
```

1.3 Preprocess Data

[9]:
```
data_c=data.cache()
```

[10]:
```
#Binarization is the process of thresholding numerical features to binary (0/1)␣
↪features.
#Binarizer takes a cvector or double, therefore casting EDUC column as␣
↪doubletype to binarize as label
data_c = data_c.withColumn("EDUC",col("EDUC").cast(DoubleType()))
binarizer = Binarizer(threshold=6.0, inputCol="EDUC", outputCol="label")
data_label = binarizer.transform(data_c)
data_label_c = data_label.cache()
```

[11]:
```
#checkign resuklts for sanity that EDUC is binarized accurately as 1 if >6 and␣
↪otherwise 0.
data_label_c.select(['EDUC','label']).distinct().sort('EDUC').show()
```

```
+----+-----+
|EDUC|label|
+----+-----+
| 0.0|  0.0|
| 1.0|  0.0|
| 2.0|  0.0|
| 3.0|  0.0|
| 4.0|  0.0|
| 5.0|  0.0|
| 6.0|  0.0|
| 7.0|  1.0|
| 8.0|  1.0|
|10.0|  1.0|
|11.0|  1.0|
+----+-----+
```

[12]:
```
#check the count for EDUC>6 or verify if flag was populated correctly
data_label_c.filter(data_label.EDUC>6).count()
```

[12]: 2470127

[13]:
```
#Verify the flag count. Should match number above
data_label_c.filter(data_label.label!=0).count()
```

[13]: 2470127

1.4 EDA

1.4.1 Full Data EDA

[14]:
```
#exploring shape of data which is number of rows and columns in the data
print("Rows, Columns = ",(data_c.count(), len(data_c.columns)))
```

```
Rows, Columns =  (5965249, 137)
```

[15]:
```
#exploring number of years in the data set
data_c.select('MULTYEAR').distinct().sort('MULTYEAR').show()
```

```
+---------+
|MULTYEAR|
+---------+
|    2015|
|    2016|
|    2017|
|    2018|
|    2019|
+---------+
```

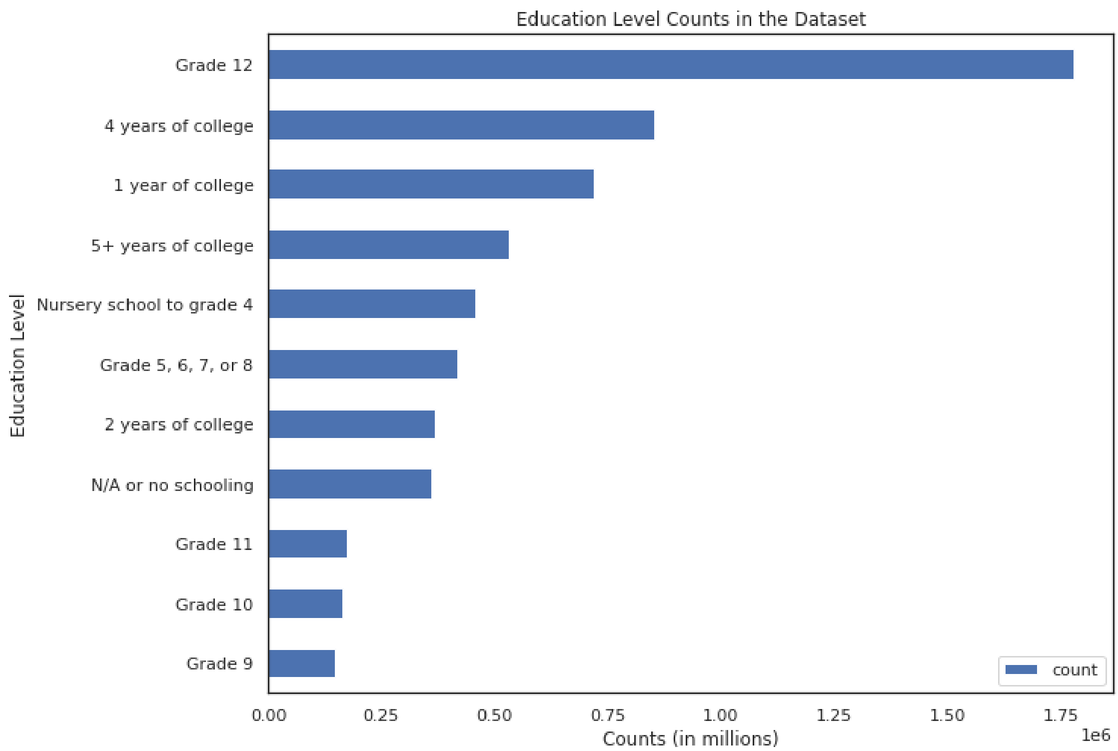
1.4.2 Education EDA

[ ]:
```
'''
Education field - we made a binary variable (above 6, and below)
EDUC                  Educational attainment [general version]
00                    N/A or no schooling
01                    Nursery school to grade 4
02                    Grade 5, 6, 7, or 8
03                    Grade 9
04                    Grade 10
05                    Grade 11
06                    Grade 12
07                    1 year of college
08                    2 years of college
09                    3 years of college
```

```
10                    4 years of college
11                    5+ years of college
'''
```

[17]:

```
#create a map to pass to yticks to have readable lables
educ_map = {0:'N/A or no schooling',\
1:'Nursery school to grade 4',\
2:'Grade 5, 6, 7, or 8',\
3:'Grade 9',\
4:'Grade 10',\
5:'Grade 11',\
6:'Grade 12',\
7:'1 year of college',\
8:'2 years of college',\
10:'4 years of college',\
11:'5+ years of college'}
#group by EDUC and and check the proportion in the datset for each level.
data_edu = data.groupby('EDUC').count()
data_edu_c = data_edu.cache()
#convert to pandas dataframe to visualize
data_edu_c = data_edu_c.toPandas()
#Add labels column for better readability
data_edu_c['Education Level'] = data_edu_c['EDUC'].map(educ_map)
#barplot is more redable when it is sorted in order of counts
data_edu_c  = data_edu_c.sort_values(by='count')
#create the barplot
data_edu_c.plot(x ='Education Level', y="count", kind='barh')
#set title, labels
plt.title("Education Level Counts in the Dataset")
plt.xlabel("Counts (in millions)")
plt.ylabel("Education Level")
plt.show() # no 9's!
```



1.4.3 Gender EDA

[ ]:
```
'''
SEX                     Sex
1                       Male
2                       Female

label
0                     Not Educated
1                     Educated
'''
```

[19]:
```
#exploring count by SEX to see if there is an imbalance in the population
data_c.groupBy('SEX').count().show()
```

```
+---+-------+
|SEX|  count|
+---+-------+
|  1|2897686|
|  2|3067563|
+---+-------+
```

[20]:
```
#exploring counts of label by SEX
data_label_c.groupBy(['label','SEX']).count().show()
```

```
+-----+---+-------+
|label|SEX|  count|
+-----+---+-------+
|  1.0|  2|1328613|
|  0.0|  1|1756172|
|  1.0|  1|1141514|
|  0.0|  2|1738950|
+-----+---+-------+
```

[21]:
```
#exploring counts of label by SEX and calculating proprotions of educated vs␣
↪not educated
male_0 = data_label_c[(data_label_c['SEX'] == 1) & (data_label_c['label'] ==␣
↪0)].count()
male_1 = data_label_c[(data_label_c['SEX'] == 1) & (data_label_c['label'] ==␣
↪1)].count()
female_0 = data_label_c[(data_label_c['SEX'] == 2) & (data_label_c['label'] ==␣
↪0)].count()
female_1 = data_label_c[(data_label_c['SEX'] == 2) & (data_label_c['label'] ==␣
↪1)].count()
male = male_0 + male_1
female = female_0 + female_1
total = male + female
print("Total (male+female) = ",total,"| Rows = ", data.count())
```

```
Total (male+female) =  5965249 | Rows =  5965249
```

[22]:
```
# Male - Ed 0 vs Ed 1 proprotions
print("Proportion of Male Not Educated (%): ",round((male_0/
↪(male_0+male_1))*100,2),\
      " | Proportion of Male Educated (%): ", round((male_1/
↪(male_0+male_1))*100,2))
```

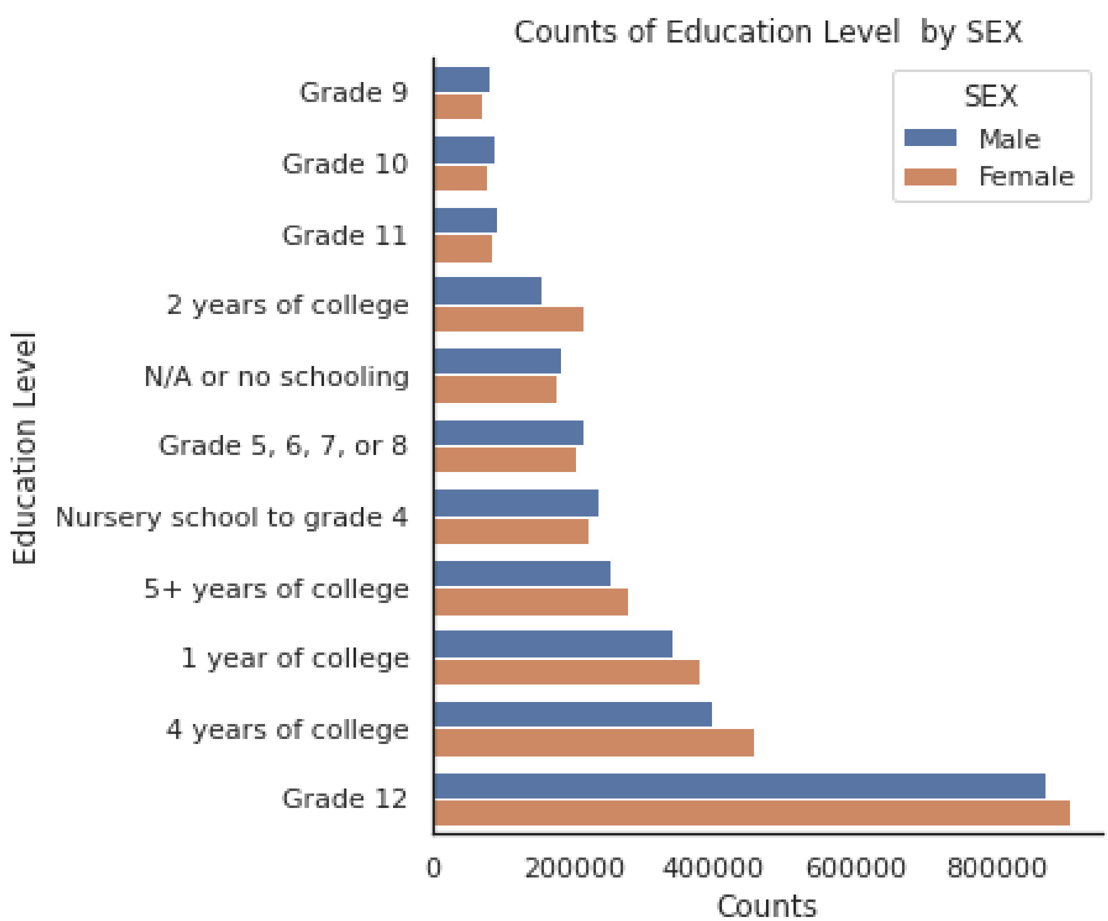
```
Proportion of Male Not Educated (%):  60.61  | Proportion of Male Educated (%):
39.39
```

[23]:
```
# Female - Ed 0 vs Ed 1 proprotions
print("Proportion of Female Not Educated (%): ",round((female_0/
↪(female_0+female_1))*100,2),\
      " | Proportion of Female Educated (%): ",round((female_1/
↪(female_0+female_1))*100,2))
```

```
Proportion of Female Not Educated (%):  56.69  | Proportion of Female Educated
(%):  43.31
```

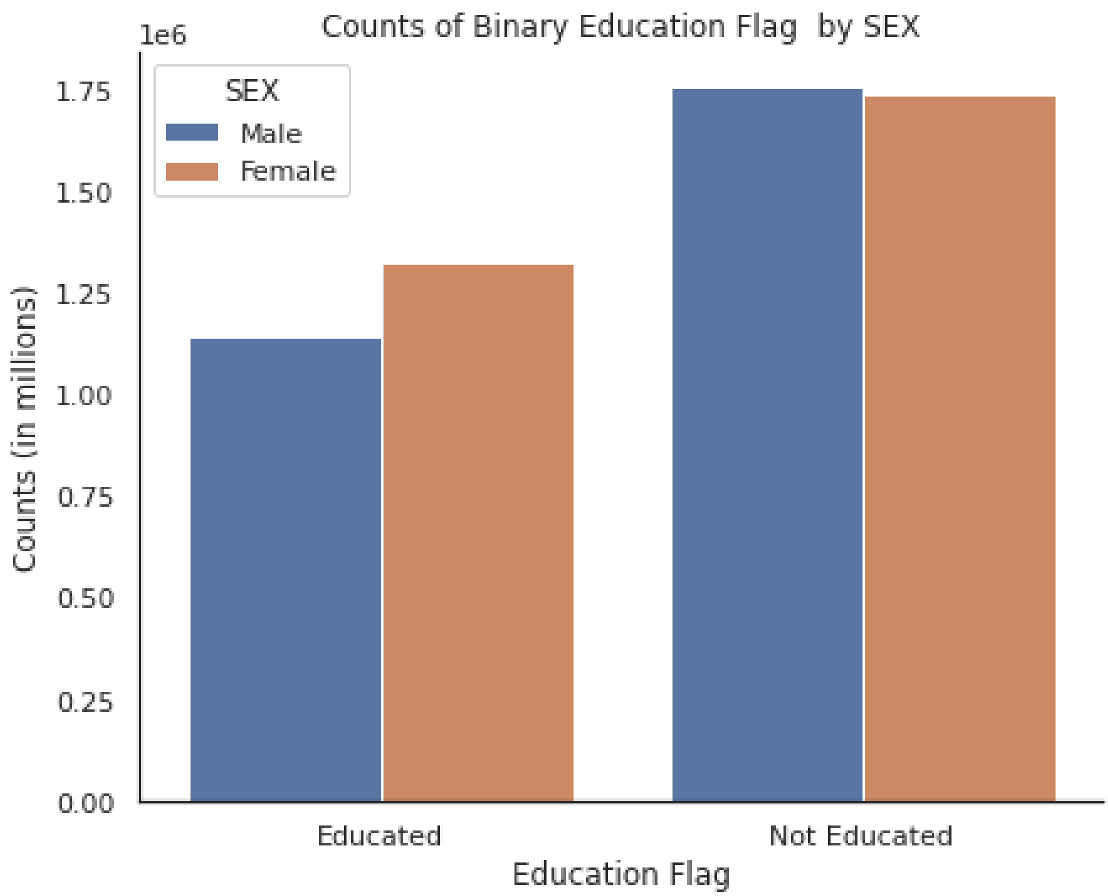
[25]:

```
#Comparing education level counts against male and female
#group by educ and sex
df2 = data.groupBy('EDUC', 'SEX').count()#.orderBy('EDUC', 'SEX')
df2_c = df2.cache()
#convert to pandas dataframe for visualizing
df2_c = df2_c.toPandas()
#add meaningful labels for EDUC
df2_c['Education Level'] = df2_c['EDUC'].map(educ_map)
#barplot is more redable when it is sorted in order of counts
df2_c  = df2_c.sort_values(by='count')
#create the plot using seaborn library
fg = sns.catplot(x='count', y='Education Level', hue='SEX', data=df2_c,␣
 ↪kind='bar',orient='h', legend_out=False,aspect=10/8);
#set x and y labels and also labels for the legend
fg.set_xlabels('Counts');
fg.set_ylabels('Education Level');
new_labels = ['Male', 'Female']
for old, new in zip(fg._legend.texts, new_labels): old.set_text(new);
#set title of the plot
plt.title('Counts of Education Level  by SEX');
```



[27]:

```
#Comparing education flag counts against male and female
#group by education flag(label) and sex
df3 = data_label.groupBy('label', 'SEX').count().orderBy('label', 'SEX')
df3_c = df3.cache()
#convert to pandas dataframe
df3_c = df3_c.toPandas()
#add meaningful labels to education flag
df3_c['Education Flag'] = df3_c['label'].map({0:'Not Educated',1:'Educated'})
#barplot is more redable when it is sorted in order of counts
df3_c  = df3_c.sort_values(by='count')
#create the plot usinmg seaborn library
fg = sns.catplot(x='Education Flag', y='count', hue='SEX', data=df3_c,␣
↪kind='bar', legend_out=False,aspect=10/8)
#set x and y labels
plt.xlabel('Education Flag')
plt.ylabel('Counts (in millions)')
#set labels for legend
new_labels = ['Male', 'Female']
for old, new in zip(fg._legend.texts, new_labels): old.set_text(new)
#set plot title
plt.title('Counts of Binary Education Flag  by SEX');
```



1.4.4 Balance the data for similar number of EDUC FLAG

[28]:
```
#majority sample is for larger class when we use a ratio by sampling smaller␣
↪class count out of larger class count
sampled_majority_df = data_label.filter(data_label['label']==0)\
    .sample(False,data_label.filter(data_label['label']==1).count()/data_label.
↪filter(data_label['label']==0).count(), seed=seed)
#minor sample is kept as is
minor_df = data_label.filter(data_label['label']==1).sample(False,1.0,␣
↪seed=seed)
#combine both in a dataframe for a balanced sample
df = sampled_majority_df.unionAll(minor_df)
df_c = df.cache()
```

[29]:
```
#check results
df_c.groupBy('label').count().show()
```

```
+-----+-------+
|label|  count|
```

```
+------+-------+
|  0.0|2471011|
|  1.0|2470127|
+------+-------+
```

1.5 EDA On Sampled Data

[30]:
```python
def createSampleData(df,cols,sampleweight):
    '''
    function to create a sample of the data based on certain columns and a␣
 ↪sample weight
    '''
    df_small = df.select(cols)
    sampled = df_small.sampleBy("MULTYEAR", fractions={2015:sampleweight, 2016:␣
 ↪sampleweight, 2017:sampleweight, 2018:sampleweight, 2019:sampleweight},␣
 ↪seed=seed)
    return sampled
```

[31]:
```python
hhtype_groups = df.groupBy("HHTYPE").count().sort(col("count").desc())
```

[32]:
```python
#udf to map hhtype
def mapHhtype(value):
    hhtype_dict = {0:'N/A',\
            1: 'Married-couple family household',\
            2: 'Male householder, no wife present',\
            3: 'Female householder, no husband present',\
            4: 'Male householder, living alone',\
            5: 'Male householder, not living alone',\
            6: 'Female householder, living alone',\
            7: 'Female householder, not living alone',\
            9: 'HHTYPE could not be determined'}
    return hhtype_dict.get(value)
```

[33]:
```python
hhtype_function = F.udf(mapHhtype, StringType())
hhtype_groups = hhtype_groups.withColumn("Household Type",␣
 ↪hhtype_function("hhtype"))
hhtype_df = hhtype_groups.toPandas()
```

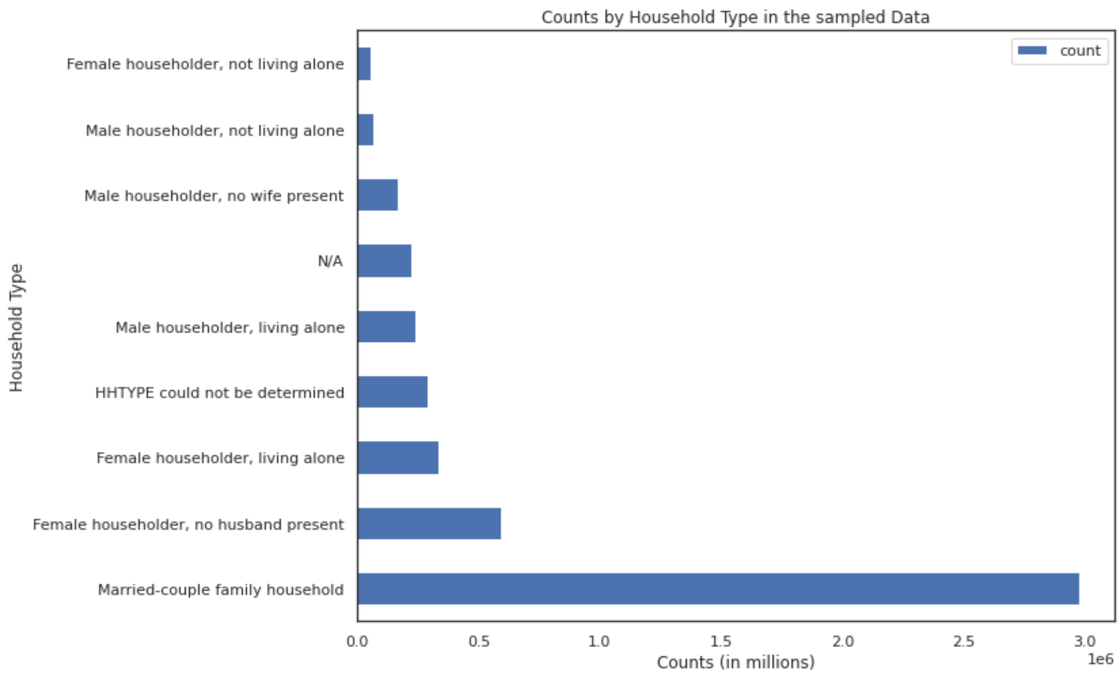
[34]:
```python
hhtype_df
```

[34]:

	HHTYPE	count	Household Type
0	1	2976745	Married-couple family household
1	3	592841	Female householder, no husband present
2	6	334681	Female householder, living alone
3	9	287870	HHTYPE could not be determined
4	4	240298	Male householder, living alone

```
5        0   220397                                    N/A
6        2   165945       Male householder, no wife present
7        5    66458      Male householder, not living alone
8        7    55903    Female householder, not living alone
```

[35]:
```
hhtype_df.plot.barh(x='Household Type', y='count', title = "House Hold Types␣
 ↪Ordered by Count");
plt.ylabel('Household Type');
plt.xlabel('Counts (in millions)');
plt.title('Counts by Household Type in the sampled Data');
```



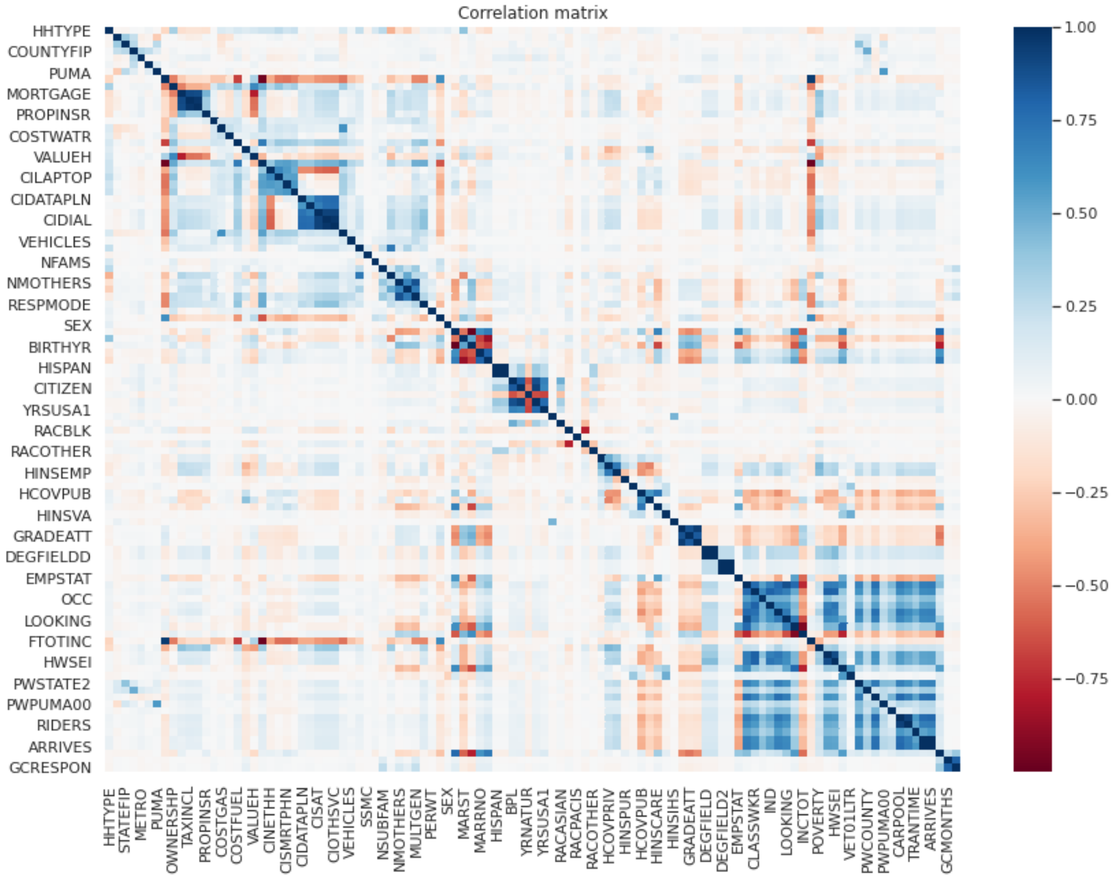
[36]:
```
#removing cols which are repeated and may be highly correlated so that PCA␣
 ↪components are more meaningful
#the columns were removed in iteration process of looking at the correlation␣
 ↪matirx below multiple times.
cols = df.drop('_c0','CLUSTER','CBSERIAL','STRATA','HHWT','EDUCD',\
 'INCSUPP',\
 'INCWAGE',\
 'INCBUS00',\
 'INCSS',\
 'INCWELFR',\
 'INCINVST',\
 'INCRETIR',\
 'INCSUPP',\
 'INCOTHER',\
```

```
 'INCEARN',\
 'RACE',\
 'RACED',\
 'SEI',\
 'CLASSWKRD',\
 'GRADEATTD',\
 'EMPSTATD',\
 'MULTGEND',\
 'OWNERSHPD',\
 'BPLD',\
 'VETSTATD',\
 'YEAR','SAMPLE','SERIAL','PERNUM','MULTYEAR','RELATED','EDUC').columns
```

[37]:
```
#creating sampled dataset (sampled)
selected_cols=[cols for cols in cols if cols not in['label']]
sampled = createSampleData(df,cols,0.1)
sampled_c = sampled.cache()
```

[38]:
```
corr_df = sampled_c.toPandas()[selected_cols]
```

[39]:
```
#View Correlation
#displaying only limited columns
corr = corr_df.corr()
plt.figure(figsize = (14, 10))
sns.heatmap(corr, cmap="RdBu",annot = False)
            #xticklabels=corr.columns.values,
            #yticklabels=corr.columns.values)
plt.title('Correlation matrix')
plt.show()
```



1.6 Model Construction

[40]:
```
split_ratio = [0.7,0.3]
folds = 3
threads = 8
evaluator = BinaryClassificationEvaluator(metricName='areaUnderPR')
```

1.6.1 Baseline Logistic Regression Model With Only 36 Features

[41]:
```
#Select Only These 36 Features for Baseline Model (roi = rows of interest)
roi =␣
↪["HHTYPE","REGION","STATEFIP","COUNTYFIP","METRO","COSTELEC","COSTGAS","COSTWATR","COSTFUEL
    "CINETHH","CILAPTOP",␣
↪"CISMRTPHN","CITABLET","VEHICLES","COUPLETYPE","NFAMS","NMOTHERS","NFATHERS",
    "CITIZEN","YRSUSA1","RACAMIND","RACASIAN","RACBLK","RACPACIS"␣
↪,"RACWHT","RACOTHER","HCOVANY","EMPSTAT",
    "LABFORCE","CLASSWKR","UHRSWORK","VETSTAT","TRANWORK","GCHOUSE","label"]
```

[42]:
```
sampled_roi = sampled.select(roi)
```

[43]:
```
#Assemble Features using VectorAssembler
baseline_assembler = VectorAssembler(inputCols=[column for column in roi if␣
↪column not in ["label"]], outputCol="features")
baseline_tr = baseline_assembler.transform(sampled_roi)

#train test split
baseline_training, baseline_test = baseline_tr.randomSplit(split_ratio,␣
↪seed=seed)

#declare model
lr_baseline = LogisticRegression(labelCol='label',
                        featuresCol='features',
                        maxIter=10,
                        regParam=0.1,
                        elasticNetParam=0.8)

#Fit model
lrModel_baseline = lr_baseline.fit(baseline_training)

#Predict on test data
lrPred_baseline = lrModel_baseline.transform(baseline_test)
lrPred_baseline_c = lrPred_baseline.cache()
```

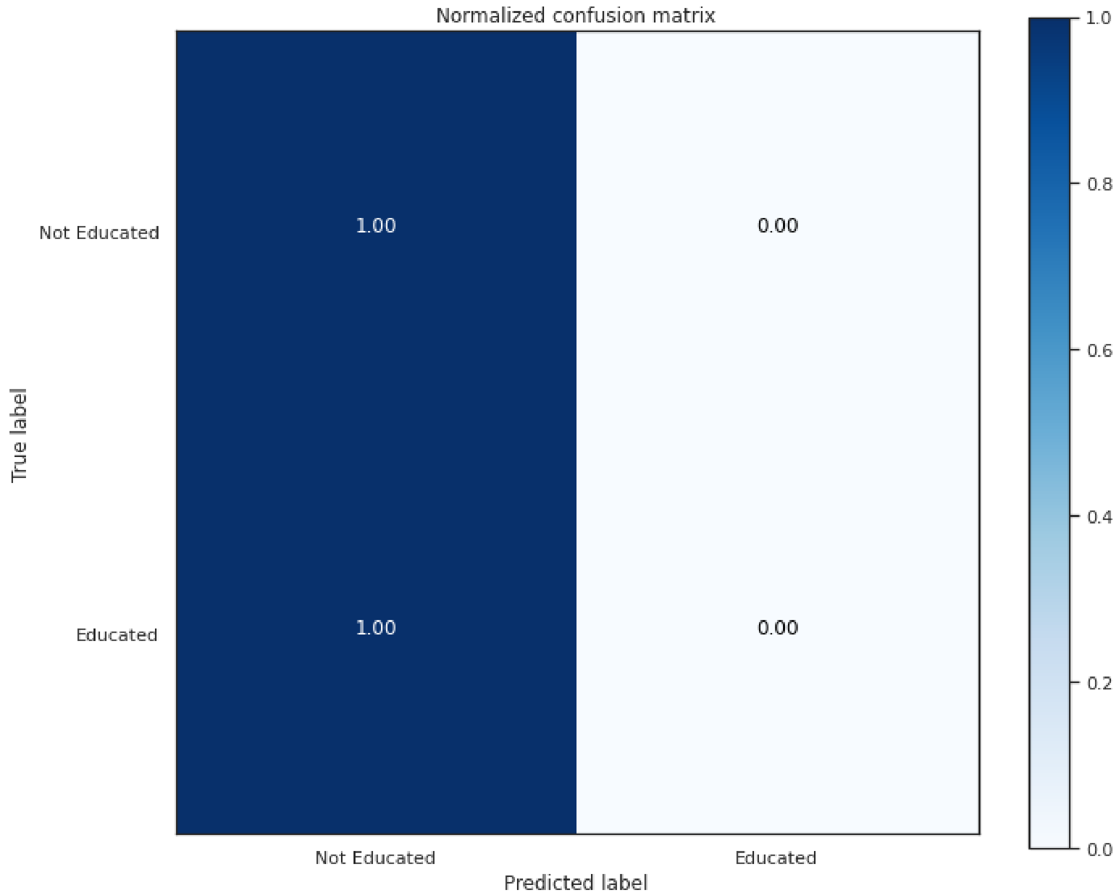
1.6.2 Evaluate Baseline Model

[44]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
```

```
                  horizontalalignment="center",
                  color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

[45]:
```
def createLabelsCMandPlot(preds):
    ##saving labels in a list to pass to the plot
    class_temp = preds.select("label").groupBy("label")\
                            .count()
    class_temp_cache = class_temp.cache()
    class_temp_cache = class_temp_cache.toPandas()
    class_temp_cache = class_temp_cache["label"].values.tolist()
    y_true = preds.select("label")
    y_true_c = y_true.cache()
    y_true_c = y_true_c.toPandas()
    y_pred = preds.select("prediction")
    y_pred_c = y_pred.cache()
    y_pred_c = y_pred_c.toPandas()

    plot_confusion_matrix(confusion_matrix(y_true_c,␣
 ↪y_pred_c,class_temp_cache), classes=['Not Educated','Educated'],␣
 ↪normalize=True,
                      title='Normalized confusion matrix')
```

[46]:
```
#call the function to evaluate and produce confusion matrix
createLabelsCMandPlot(lrPred_baseline_c)
```



[47]:
```
def classificationMetrics(preds):
    #calcualte classification report
    TN = preds.filter('prediction = 0 AND label = prediction').count()
    TP = preds.filter('prediction = 1 AND label = prediction').count()
    FN = preds.filter('prediction = 0 AND label <> prediction').count()
    FP = preds.filter('prediction = 1 AND label <> prediction').count()
    # show confusion matrix
    preds.groupBy('label', 'prediction').count().show()
    preds_c = preds.cache()
    # calculate metrics by the confusion matrix
    accuracy = (TN + TP) / (TN + TP + FN+ FP)
    if TP + FP == 0:
        precision = 0
    else:
        precision = TP / (TP + FP)
    if TP + FN == 0:
        recall = 0
    else:
        recall = TP / (TP + FN)
```

```
    if precision + recall == 0:
        F = 0
    else:
        F =  2 * (precision*recall) / (precision + recall)
    # calculate auc
    evaluator = BinaryClassificationEvaluator(rawPredictionCol="prediction",
                                              labelCol="label",
                                              metricName="areaUnderPR")
    auc = evaluator.evaluate(preds_c)
    print('precision: %0.3f' % precision)
    print('recall: %0.3f' % recall)
    print('accuracy: %0.3f' % accuracy)
    print('F1 score: %0.3f' % F)
    print('AUC: %0.3f' % auc)
```

[48]:
```
#call the function to display classification metrics
classificationMetrics(lrPred_baseline)
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       0.0|73853|
|  0.0|       0.0|73165|
+-----+----------+-----+

precision: 0.000
recall: 0.000
accuracy: 0.498
F1 score: 0.000
AUC: 0.502
```

1.6.3 Vector Assemble all features

[49]:
```
#pass all the features into vector assembler to create a vector format to pass␣
↪tto the classification model
assembler = VectorAssembler(inputCols=[cols for cols in cols if cols!='label'],␣
↪outputCol="features")
transformed = assembler.transform(sampled)

#register table as sql table and keep only columns of interest and save in a␣
↪new dataframe. This can be done without using SQl as well.
transformed.registerTempTable('transformed_tbl')
transformed_df = sqlContext.sql('select label,features from transformed_tbl')
transformed_df_c = transformed_df.cache()
transformed_df_c.show(5)
```

```
+-----+--------------------+
```

```
|label|            features|
+-----+--------------------+
|  0.0|(106,[0,1,2,6,7,8…|
|  0.0|[9.0,32.0,1.0,0.0…|
|  0.0|(106,[0,1,2,6,7,8…|
|  0.0|(106,[0,1,2,3,4,6…|
|  0.0|[4.0,32.0,1.0,0.0…|
+-----+--------------------+
only showing top 5 rows
```

1.6.4 Split Data Into Train Test Split

[50]:
```
def splitData(dataframe,split_ratio,seed):
    '''
    function to split the data into train and test, and cache the resulting␣
 ↪dataframe
    '''
    training_data, test_data = dataframe.randomSplit(split_ratio, seed=seed)
    cached_tr = training_data.cache()
    cached_test = test_data.cache()
    return cached_tr,cached_test
```

[51]:
```
#train test split on sampled data
training_data, test_data = splitData(transformed_df_c,split_ratio,seed)
```

1.6.5 Scale Data to Prepare for PCA

[52]:
```
#scale the training data to use in pipeline
scaler_train = StandardScaler(inputCol="features", outputCol="scaledFeatures")
```

1.6.6 PCA

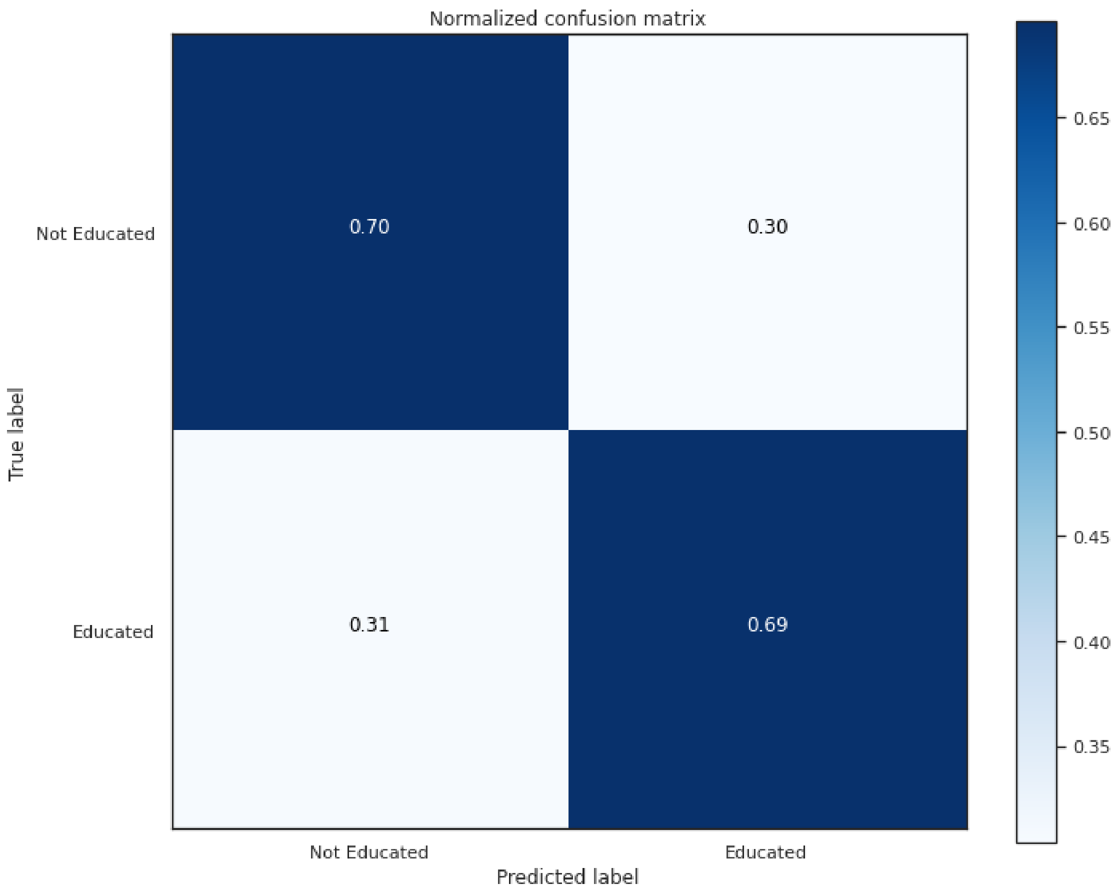
[53]:
```
#pca to reduce 100 odd features into principal components - on training data␣
 ↪only because that is our model
pca_model = PCA(k=10, inputCol = "scaledFeatures", outputCol = "pca_features")
```

1.6.7 Logistic Regression Pipeline and Evaluation

[54]:
```
#create a logistic regression model to pass into pipeline
lr = LogisticRegression(labelCol='label',
                        featuresCol='pca_features',
                        maxIter=10,
                        regParam=0.1,
                        elasticNetParam=0.8)
```

```
#creating a pipeline with the pca and model to use
lr_pipeline = Pipeline(stages = [scaler_train, pca_model, lr])

lr_model = lr_pipeline.fit(training_data)
lrPred = lr_model.transform(test_data)
lrPred_c = lrPred.cache()
```

[55]:
```
createLabelsCMandPlot(lrPred_c)
```



[56]:
```
classificationMetrics(lrPred_c)
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0|51204|
|  0.0|       1.0|22233|
|  1.0|       0.0|22649|
|  0.0|       0.0|50932|
+-----+----------+-----+
```

```
precision: 0.697
recall: 0.693
accuracy: 0.695
F1 score: 0.695
AUC: 0.667
```

1.6.8 Models

[57]:
```
#create a SVM classifier model to pass into pipeline
lsvc = LinearSVC(labelCol = "label", featuresCol = "pca_features", maxIter=10,␣
 ↪regParam=0.1)

#create a Gradient Boosting classifier model to pass into pipeline
gb = GBTClassifier(labelCol = "label", featuresCol = "pca_features",␣
 ↪maxDepth=10)

#create a Random Forest classifier model to pass into pipeline
rf = RandomForestClassifier(labelCol = "label", featuresCol = "pca_features",␣
 ↪maxDepth=15)
```

1.6.9 Param Grids

[99]:
```
#SVM param grid
paramGrid_svm = ParamGridBuilder() \
  .addGrid(lsvc.regParam, [0.1,0.5,1]) \
  .build()

#Gradient param grid
paramGrid_gbt = ParamGridBuilder() \
             .addGrid(gb.maxIter, [10, 15]) \
             .addGrid(gb.maxBins, [10, 20]) \
             .build()

#.addGrid(gb.maxIter, [10, 15]) \

#Random Forest param grid
paramGrid_rf = ParamGridBuilder() \
        .addGrid(rf.numTrees, [20, 30]) \
        .build()
```

[100]:
```
def␣
 ↪modelTrainingCrossVal(model,paramGrid,evaluator,folds,seed,threads,scaler_train,␣
 ↪pca_model, train_data):
    '''
```

```
    This function creates a cross validator. Cross vlaidator takes a model as␣
↪an estimator, param grid for a model, evaluator, number of folds, seed and␣
↪threads.
    Creates a pipeline and passes scaled data, pca model and cross validator to␣
↪a pipeline. It saves time by passing cross validator into pipeline,
    instead of piepeline as an estimator. Otherwise it build a piepeline for␣
↪each model the cross validator evaluates. Which is not efficient in this␣
↪case.
    '''
    crossval = CrossValidator(estimator = model,\
                              estimatorParamMaps=paramGrid,\
                              evaluator = evaluator,\
                              numFolds= folds,seed=seed,
                              parallelism=threads)

    #creating a pipeline with the pca and model to use in the cross validator
    pipeline = Pipeline(stages = [scaler_train, pca_model, crossval])
    pipeline_m = pipeline.fit(train_data)
    return pipeline_m

def evalModelandTrainingMetrics(model):
    print("Training Model AUCROC from Cross-validation:",model.avgMetrics)
```

1.6.10 Support Vector Machine Pipeline, Model, Cross Validation and Evaluation

[ ]:
```
#train model
svm_pipe_m =␣
↪modelTrainingCrossVal(lsvc,paramGrid_svm,evaluator,folds,seed,threads,scaler_train,␣
↪pca_model, training_data)

#Evaluate Model
evalModelandTrainingMetrics(svm_pipe_m.stages[2])
```

[ ]:
```
#Get the hyperparameters from svm model
print("RegParam parameter: {}".format(svm_pipe_m.stages[2].bestModel.
↪getRegParam()))
```

[ ]:
```
#predict and evaluate the svm model
svmPred = svm_pipe_m.transform(test_data)
svmPred_c = svmPred.cache()
```

[ ]:
```
#plot confusion matrix
createLabelsCMandPlot(svmPred_c)
```

[98]:
```
#print classification metrics
print("SVM Classification Metrics for Test Data:")
```

```
print(" ")
classificationMetrics(svmPred_c)
```

```
SVM Classification Metrics for Test Data:

+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       0.0|73853|
|  0.0|       0.0|73165|
+-----+----------+-----+

precision: 0.000
recall: 0.000
accuracy: 0.498
F1 score: 0.000
AUC: 0.502
```

1.6.11 Gradient Boosting Pipeline, Model, Cross Validation and Evaluation

[65]:
```
#train model
gb_pipe_m =␣
↪modelTrainingCrossVal(gb,paramGrid_gbt,evaluator,folds,seed,threads,scaler_train,␣
↪pca_model, training_data)
```

[66]:
```
#Evaluate Model
evalModelandTrainingMetrics(gb_pipe_m.stages[2])
```

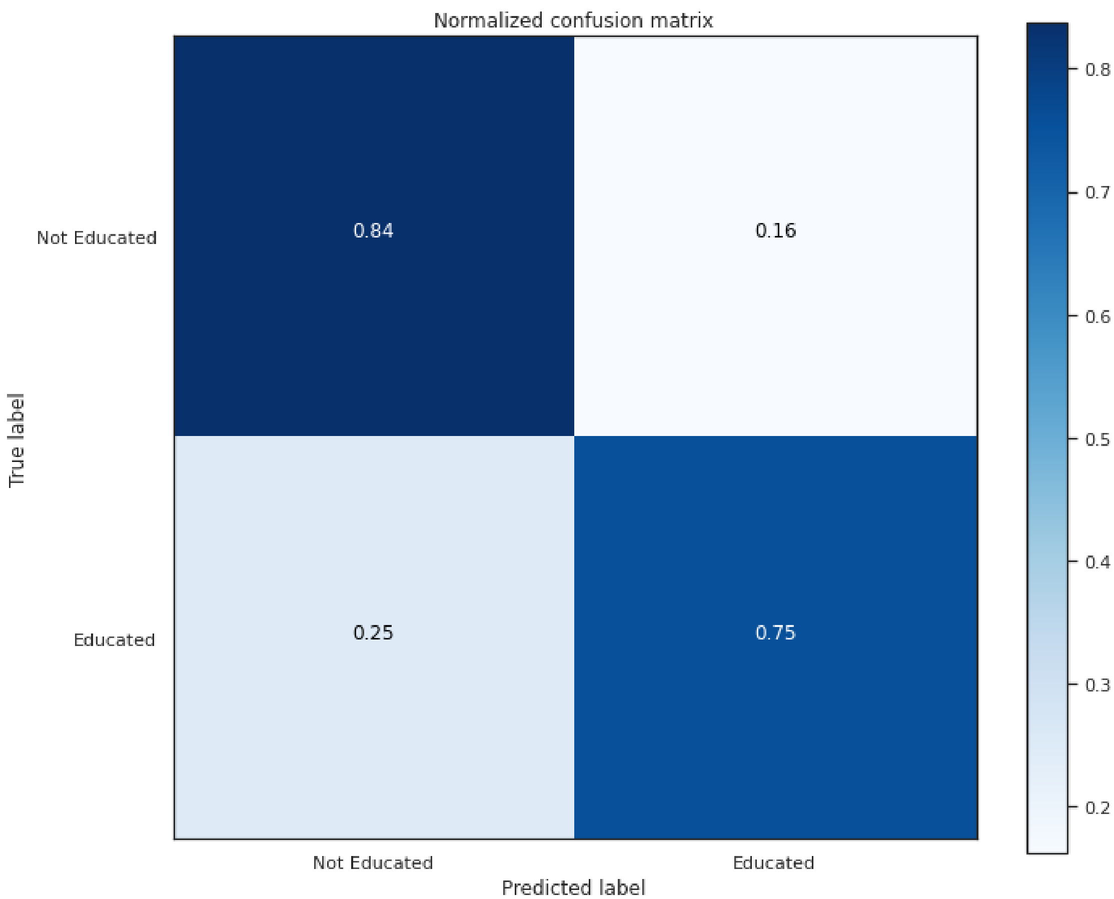
```
Training Model AUCROC from Cross-validation: [0.9054715507315008,
0.9064082868974739, 0.9069456783009662, 0.9081776946253195]
```

[67]:
```
#Get the hyperparameters from the pca and for the GB model
cv_gb_model = gb_pipe_m.stages[2].bestModel

#GB parameters
print("Max Iter parameter: {}".format(cv_gb_model.getMaxIter()))
print("Max bins parameter: {}".format(cv_gb_model.getMaxBins()))
```

```
Max Iter parameter: 15
Max bins parameter: 20
```

[68]:
```
#predict and evaluate the gb model
gbPred = gb_pipe_m.transform(test_data)
gbPred_c = gbPred.cache()
```

[69]:
```
#plot confusion matrix
createLabelsCMandPlot(gbPred_c)
```



[70]:
```
#print classification metrics
print("GBT Classification Metrics for Test Data:")
print(" ")
classificationMetrics(gbPred_c)
```

```
GBT Classification Metrics for Test Data:

+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0|55644|
|  0.0|       1.0|11862|
|  1.0|       0.0|18209|
|  0.0|       0.0|61303|
+-----+----------+-----+

precision: 0.824
recall: 0.753
accuracy: 0.795
```

```
F1 score: 0.787
AUC: 0.785
```

1.6.12 Random Forest Pipeline, Model, Cross Validation and Evaluation

[71]:
```
#train model
rf_pipe_m = modelTrainingCrossVal(rf,paramGrid_rf,evaluator,folds,seed,threads,scaler_train, pca_model, training_data)
```

[72]:
```
#Evaluate Model
evalModelandTrainingMetrics(rf_pipe_m.stages[2])
```

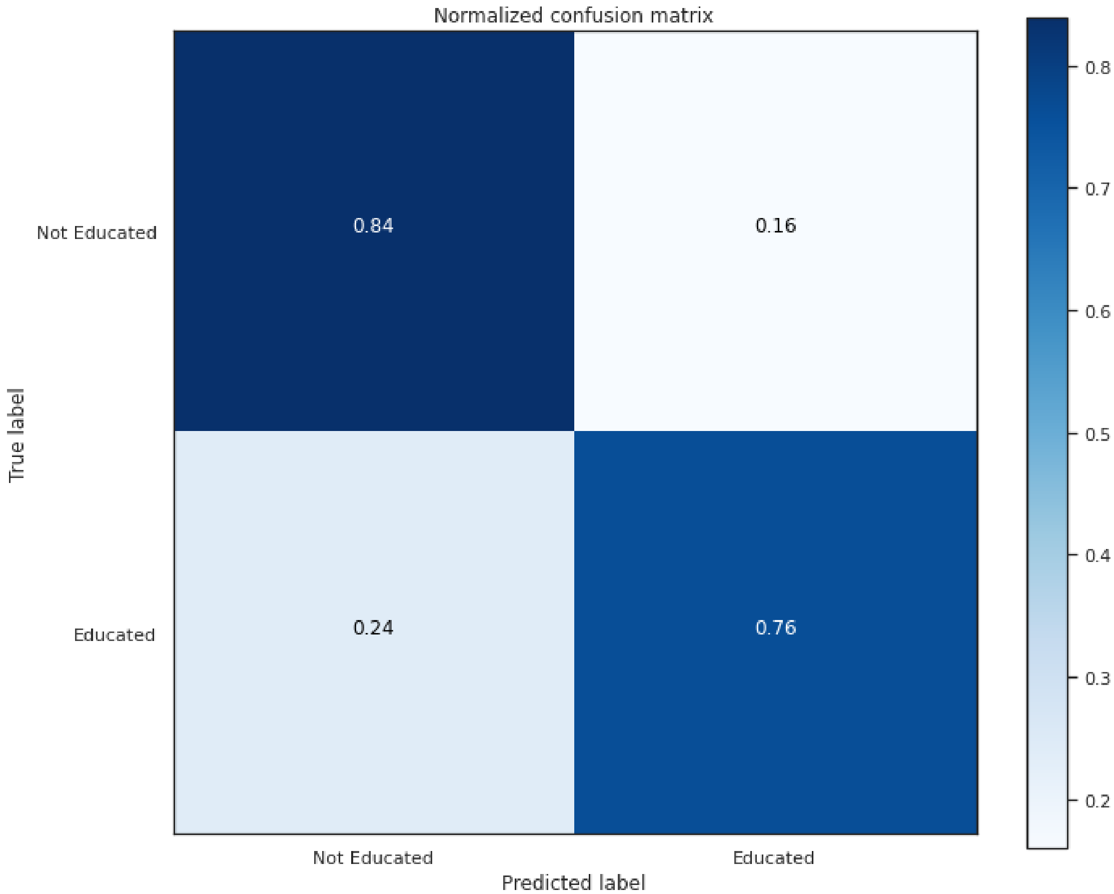
```
Training Model AUCROC from Cross-validation: [0.912075695322395,
0.9130149163898613]
```

[73]:
```
#Get the hyperparameters from the pca and for the RF model
rf_model = rf_pipe_m.stages[2].bestModel

#RF parameters
print("Num Trees parameter: {}".format(rf_model.explainParam('numTrees')))
```

```
Num Trees parameter: numTrees: Number of trees to train (>= 1). (default: 20,
current: 30)
```

[74]:
```
#predict and evaluate the rf model
rfPred = rf_pipe_m.transform(test_data)
rfPred_c = rfPred.cache()
```

[75]:
```
#plot confusion matrix
createLabelsCMandPlot(rfPred_c)
```



[76]:
```
#print classification metrics
print("RF Classification Metrics for Test Data:")
print(" ")
classificationMetrics(rfPred_c)
```

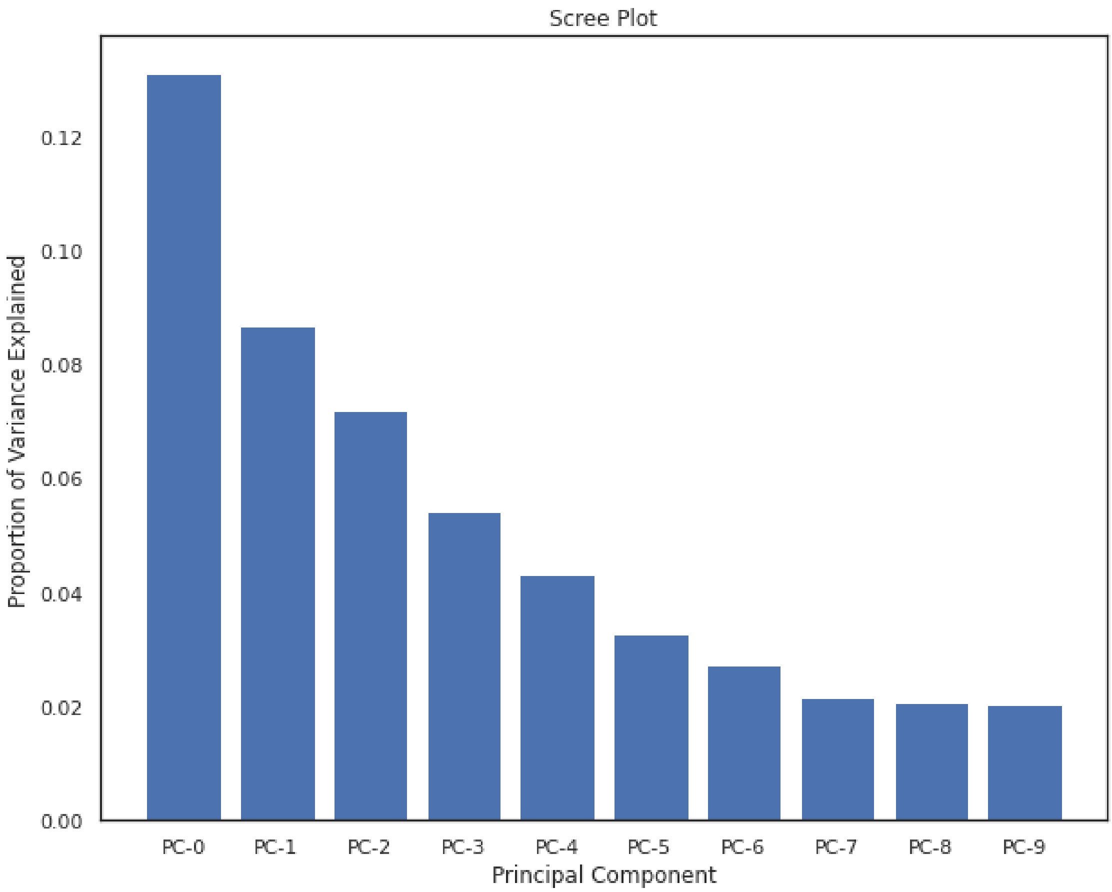
```
RF Classification Metrics for Test Data:

+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0|56267|
|  0.0|       1.0|11710|
|  1.0|       0.0|17586|
|  0.0|       0.0|61455|
+-----+----------+-----+

precision: 0.828
recall: 0.762
accuracy: 0.801
```

```
F1 score: 0.793
AUC: 0.789
```

1.6.13 Interpretation of PCA

[77]:
```
##PCA Loadings
pipe = rf_pipe_m.stages[1]
exp_var = pipe.explainedVariance
print("Explained Variance: ",exp_var)
rows = pipe.pc.toArray().tolist()
pca_components=['PC-0','PC-1','PC-2','PC-3','PC-4','PC-5','PC-6','PC-7','PC-8','PC-9']#,\
 ↪#'PC-11','PC-12','PC-13','PC-14','PC-15','PC-16','PC-17','PC-18','PC-19']
#screeplot
plt.bar(pca_components,exp_var)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.show()
```

```
Explained Variance:  [0.1309759597302075,0.08684184800315224,0.07198521655484985
,0.054104316748693475,0.043177461690147044,0.032747564586555145,0.02728318428534
4698,0.021730785563836088,0.020756421482399815,0.020269617575580077]
```



[78]:

```
#create a spark dataframe with pca loadings, componetns and column names for
↪further analysis
df_pca = spark.createDataFrame(rows,pca_components)
df_pandas = df_pca.toPandas()
df_pandas.index = selected_cols
df_pandas.sort_values(by='PC-0', ascending=False)
```

[78]:

```
                PC-0      PC-1      PC-2      PC-3      PC-4      PC-5  \
INCTOT      0.158591  0.184800  0.049826 -0.016828 -0.016424 -0.044083
MARST       0.146909  0.069207  0.151275  0.092781 -0.069195  0.038329
HINSCAID    0.118633  0.000781 -0.003271  0.100496 -0.015888  0.031373
HCOVPUB     0.107334 -0.098972 -0.173188  0.032390 -0.016094  0.064111
SCHOOL      0.099895  0.105831  0.125517 -0.030002 -0.006904  0.000263
RELATE      0.098453 -0.084324  0.197891 -0.033684  0.029462  0.024853
SCHLTYPE    0.097520  0.101263  0.128342 -0.038972 -0.004528 -0.001334
GRADEATT    0.097489  0.098992  0.146690 -0.025978 -0.007063  0.011165
BIRTHYR     0.088255  0.204670  0.216132  0.034204 -0.024374 -0.016958
FTOTINC     0.075230 -0.202507  0.214547 -0.130315  0.046696  0.010354
GQ          0.073730 -0.200251  0.218294 -0.119966  0.042607  0.012707
```

FOODSTMP	0.068011	0.015309	-0.002390	0.159120	-0.009812	0.020291
VALUEH	0.064141	-0.074092	0.133612	0.204604	-0.017479	0.223068
HINSCARE	0.043417	-0.149394	-0.217629	-0.039162	0.005042	0.042409
EMPSTAT	0.042557	-0.200054	-0.163736	-0.055871	0.058309	0.067982
RACBLK	0.036322	-0.024435	0.046251	0.085250	-0.041389	0.015426
YRNATUR	0.035281	0.005850	0.010495	-0.057003	-0.327398	0.020057
CINETHH	0.026432	-0.029557	-0.130796	0.237822	-0.086249	-0.321829
NMOTHERS	0.024235	0.184578	0.042050	0.027906	0.066659	-0.114773
NSUBFAM	0.013800	0.057992	0.013643	0.069729	0.027432	-0.032198
RACOTHER	0.011000	0.017502	0.024129	0.064289	0.104845	-0.017784
SEX	0.009627	0.006184	-0.032348	0.012723	0.007391	0.011384
RACAMIND	0.008892	0.000898	0.005814	0.017024	-0.016311	-0.007660
HISPAN	0.008647	0.030876	0.031303	0.097295	0.216144	-0.023510
HISPAND	0.008589	0.030128	0.031131	0.095407	0.212932	-0.023021
PERWT	0.007278	0.054256	0.021899	0.099348	0.015261	-0.000559
HINSIHS	0.006993	0.001933	0.001152	0.012342	-0.015717	-0.009614
REGION	0.006757	0.005144	0.013984	0.018089	-0.012659	-0.035586
CILAPTOP	0.006317	0.052050	-0.161325	0.231591	-0.074772	-0.137112
COSTWATR	0.005986	0.013959	-0.041158	0.119050	-0.038669	0.062587
MULTGEN	0.004117	0.228177	-0.029769	0.071026	0.041160	-0.109673
NFATHERS	0.002874	0.179933	0.037472	-0.033436	0.080635	-0.140273
NFAMS	0.002062	0.004340	0.032545	0.063044	-0.013389	0.059325
RACPACIS	0.001813	0.001887	0.007030	0.006524	0.013635	0.001617
HHTYPE	-0.000204	0.006428	-0.021278	0.179552	-0.085351	0.097937
HINSTRI	-0.001910	-0.019369	-0.030680	-0.038503	-0.005276	0.031514
HINSPUR	-0.002351	-0.038611	-0.065297	-0.033792	0.008507	0.030661
OWNERSHP	-0.002820	0.100922	-0.053012	0.290115	-0.055034	0.191269
HINSVA	-0.004081	-0.047060	-0.068485	-0.021445	-0.022742	0.033033
CITY	-0.004735	-0.000671	0.017767	0.016315	0.015020	0.029504
COSTGAS	-0.007962	0.061441	-0.067590	0.133166	-0.033014	0.055788
GCMONTHS	-0.008625	-0.008123	-0.031147	0.013325	0.011976	-0.029726
CISMRTPHN	-0.008800	0.040900	-0.207521	0.156829	-0.067245	-0.104559
SSMC	-0.009390	0.001969	-0.005744	-0.010811	-0.000733	-0.000805
CITABLET	-0.009880	0.045000	-0.171221	0.185159	-0.064628	-0.064354
STATEFIP	-0.011357	0.011882	0.028165	-0.000265	0.002061	-0.025832
GCRESPON	-0.011622	-0.006637	-0.043487	0.017714	0.032596	-0.040203
RACASIAN	-0.011768	0.023990	0.021219	0.013618	0.194104	-0.000571
COSTELEC	-0.012400	0.089444	-0.062990	0.042255	-0.020590	-0.024640
HCOVANY	-0.013352	0.032015	-0.068300	-0.102350	-0.058305	0.000553
PUMA	-0.015113	0.006034	0.012153	-0.009783	0.030977	0.004289
COUNTYFIP	-0.015330	0.020171	0.027606	0.002846	0.042124	0.015458
VEHICLES	-0.017913	0.074111	-0.067844	0.070061	-0.016315	-0.072178
METRO	-0.022219	0.030099	0.016567	-0.019958	0.086079	0.036145
RACWHT	-0.027009	0.011918	-0.054948	-0.108549	-0.088148	-0.003725
VET01LTR	-0.028503	-0.058677	-0.046945	-0.033014	-0.032019	0.042235
FUELHEAT	-0.032272	0.115813	-0.135422	0.131596	-0.052866	0.016274
CITIZEN	-0.032368	-0.000667	0.013070	0.110324	0.392559	-0.015181

YRSUSA1	-0.034199	-0.018661	-0.022361	0.072808	0.358130	-0.024451
YRIMMIG	-0.036005	-0.000550	0.009622	0.107850	0.416372	-0.015922
DEGFIELD2	-0.037002	-0.002711	-0.005775	-0.031836	0.005512	0.053203
DEGFIELD2D	-0.037005	-0.002714	-0.005777	-0.031837	0.005515	0.053202
BPL	-0.037160	0.002253	0.010414	0.080059	0.387358	-0.006059
COUPLETYPE	-0.038230	0.100498	-0.033455	-0.028257	0.022949	-0.028767
PROPINSR	-0.052280	0.058312	-0.068158	-0.176398	0.034835	-0.118545
NCOUPLES	-0.055397	0.128462	-0.074669	-0.110853	0.077961	-0.098582
COSTFUEL	-0.056420	0.160555	-0.146284	0.102168	-0.018440	0.008554
CIHISPEED	-0.060076	0.158743	-0.059267	-0.063456	0.021918	0.275059
INSINCL	-0.063677	0.117281	-0.014831	-0.197961	0.020799	-0.245942
CIDATAPLN	-0.064698	0.152149	-0.077045	-0.092080	0.031963	0.278513
TAXINCL	-0.065516	0.118707	-0.013837	-0.200510	0.022333	-0.245281
RESPMODE	-0.070187	0.162243	-0.063455	-0.002063	0.005195	0.032308
MORTGAGE	-0.072758	0.114780	-0.062930	-0.226280	0.022935	-0.264129
FRIDGE	-0.076605	0.208506	-0.213496	0.122085	-0.045119	-0.011563
CISAT	-0.085628	0.195720	-0.055607	-0.114496	0.042771	0.290400
PWPUMA00	-0.088568	0.008711	0.055349	0.026170	0.000795	-0.011315
AGE	-0.088641	-0.204579	-0.216070	-0.035208	0.025015	0.018688
CIDIAL	-0.088883	0.203477	-0.059238	-0.120170	0.041153	0.296465
CIOTHSVC	-0.089427	0.203132	-0.061940	-0.121547	0.043194	0.297581
PWCOUNTY	-0.102878	0.013808	0.067308	0.036807	0.001595	-0.007314
HINSEMP	-0.103298	0.081763	0.052851	-0.111967	-0.042128	-0.059564
MARRNO	-0.105558	-0.113982	-0.171923	-0.051587	0.024107	-0.010766
HCOVPRIV	-0.108332	0.059655	0.012807	-0.146308	-0.036740	-0.034300
TRANWORK	-0.108841	-0.007855	0.067455	0.018823	-0.012405	0.005742
DEGFIELD	-0.111203	-0.011667	-0.024149	-0.081813	0.024688	0.072556
DEGFIELDD	-0.111221	-0.011683	-0.024172	-0.081828	0.024710	0.072569
GCHOUSE	-0.127416	-0.147203	-0.168660	-0.015341	0.047850	-0.019240
OCC	-0.128690	-0.030695	0.090464	0.090194	-0.032611	-0.007588
POVERTY	-0.131683	0.094399	-0.076992	-0.152214	-0.005397	-0.049904
YRMARR	-0.137625	-0.121613	-0.181037	-0.056409	0.052872	-0.024201
VETSTAT	-0.141919	-0.183258	-0.088981	-0.006174	-0.006261	0.054851
TRANTIME	-0.154880	0.020034	0.078208	0.056489	-0.025618	-0.041167
AVAILBLE	-0.170953	-0.181970	-0.041681	0.016947	0.015430	0.041982
RIDERS	-0.181548	0.022833	0.103719	0.082377	-0.037602	-0.044249
IND	-0.184622	-0.019967	0.098093	0.036985	-0.031144	0.028826
PWSTATE2	-0.189751	0.015651	0.112349	0.066657	-0.042828	-0.036284
DEPARTS	-0.190175	0.018375	0.126946	0.094173	-0.051913	-0.026277
ARRIVES	-0.193508	0.019063	0.127415	0.094155	-0.051747	-0.027941
CARPOOL	-0.201297	0.026180	0.109737	0.085961	-0.045167	-0.044755
PWTYPE	-0.202877	0.013331	0.109448	0.080618	-0.059663	-0.041355
LOOKING	-0.206596	-0.095528	0.095928	0.050537	-0.019922	-0.004086
OCCSCORE	-0.209462	-0.014709	0.075392	0.008918	-0.027117	0.013306
HWSEI	-0.209620	-0.008225	0.070526	-0.006012	-0.026739	0.019088
CLASSWKR	-0.212056	-0.022122	0.112726	0.054626	-0.040627	0.010271
UHRSWORK	-0.214546	0.005665	0.115758	0.057818	-0.040713	-0.021761

LABFORCE	-0.235907	-0.082638	0.059468	0.067254	-0.024680	0.006296

	PC-6	PC-7	PC-8	PC-9
INCTOT	0.040256	0.052985	-0.083087	-0.015721
MARST	0.096885	-0.042506	0.165895	0.044524
HINSCAID	-0.121046	0.087370	0.107192	0.015112
HCOVPUB	-0.061724	0.021100	0.077963	0.092421
SCHOOL	0.134377	-0.129577	0.072992	0.117630
RELATE	-0.087938	0.017025	0.051544	-0.019876
SCHLTYPE	0.138283	-0.126602	0.068092	0.113200
GRADEATT	0.131286	-0.121191	0.078163	0.109459
BIRTHYR	0.010306	0.027842	-0.025563	-0.038590
FTOTINC	-0.064299	0.023656	-0.012909	-0.024777
GQ	-0.053038	0.010958	-0.007499	-0.015031
FOODSTMP	-0.157689	0.099118	0.122201	0.005760
VALUEH	0.011357	0.097126	-0.113231	-0.019108
HINSCARE	0.021528	-0.034515	0.018637	0.074439
EMPSTAT	-0.029370	-0.020479	0.065418	0.018075
RACBLK	-0.041103	0.079586	0.333372	0.180000
YRNATUR	-0.077246	0.040121	-0.073692	-0.018583
CINETHH	0.146516	0.035553	-0.029052	-0.000894
NMOTHERS	-0.238158	0.173264	0.024389	0.014934
NSUBFAM	-0.234319	0.168922	0.073197	0.009758
RACOTHER	-0.036362	-0.036128	-0.060756	-0.033037
SEX	0.025042	0.043434	0.076627	-0.053549
RACAMIND	-0.037429	0.011290	0.001474	0.014240
HISPAN	-0.033137	-0.113539	-0.163487	-0.179192
HISPAND	-0.032302	-0.113311	-0.162734	-0.178542
PERWT	-0.011799	0.005102	0.050869	0.031748
HINSIHS	-0.034003	0.007368	-0.005206	0.013968
REGION	-0.050519	0.072477	-0.227816	0.130168
CILAPTOP	0.063678	-0.015866	0.002169	0.013472
COSTWATR	0.030742	-0.033621	-0.012007	-0.057497
MULTGEN	-0.212528	0.155730	0.050628	0.037695
NFATHERS	-0.200041	0.139763	-0.099780	-0.023139
NFAMS	0.038319	-0.039125	0.089794	0.020158
RACPACIS	0.004859	0.006605	0.006005	0.023274
HHTYPE	0.088312	-0.021244	0.149787	-0.002903
HINSTRI	0.014732	-0.034437	-0.038480	0.170284
HINSPUR	0.079828	-0.058018	-0.000137	0.019753
OWNERSHP	0.060325	0.068470	-0.092081	0.002697
HINSVA	-0.015920	-0.054583	-0.001966	0.187592
CITY	0.059424	0.076906	-0.005089	0.139758
COSTGAS	0.017008	-0.099632	0.055823	-0.120056
GCMONTHS	-0.215221	0.172077	0.109623	0.090602
CISMRTPHN	0.097562	-0.046562	0.015496	0.041252
SSMC	0.003213	0.009004	-0.032682	-0.024549

CITABLET	0.078992	-0.061871	0.017150	0.026883
STATEFIP	0.010338	0.122357	-0.263089	0.314615
GCRESPON	-0.266159	0.202275	0.122767	0.098869
RACASIAN	0.099063	0.062945	0.086603	0.134479
COSTELEC	-0.034908	0.021737	0.007127	0.000354
HCOVANY	0.142859	0.012762	0.020346	0.096430
PUMA	0.066121	0.062243	-0.072301	0.242915
COUNTYFIP	0.065361	0.151016	-0.187452	0.337410
VEHICLES	-0.038055	0.025163	0.034120	0.034162
METRO	0.089683	0.058994	0.008473	0.086626
RACWHT	0.013843	-0.085846	-0.312624	-0.209388
VET01LTR	-0.022019	-0.060083	-0.034973	0.215319
FUELHEAT	0.020670	-0.092264	0.032951	-0.085075
CITIZEN	0.056294	-0.038227	0.009516	-0.003496
YRSUSA1	0.066315	-0.058053	0.040840	0.008590
YRIMMIG	0.068432	-0.047008	0.024784	-0.001385
DEGFIELD2	0.183356	0.378952	0.084490	-0.188993
DEGFIELD2D	0.183363	0.378948	0.084492	-0.188989
BPL	0.101748	-0.001869	0.051249	0.075349
COUPLETYPE	-0.102421	0.061038	-0.168227	-0.090113
PROPINSR	0.045156	-0.009126	0.045752	-0.008814
NCOUPLES	-0.139844	0.075477	-0.241627	-0.071335
COSTFUEL	0.070294	0.000301	0.005740	0.027381
CIHISPEED	-0.114379	-0.075561	0.033944	0.017994
INSINCL	-0.014724	-0.072698	0.209512	0.026037
CIDATAPLN	-0.078000	-0.079874	0.049226	0.032458
TAXINCL	-0.009857	-0.070209	0.208528	0.026650
RESPMODE	0.071804	0.036137	-0.019526	0.004470
MORTGAGE	-0.013001	-0.085676	0.192052	0.023683
FRIDGE	0.066419	-0.019232	0.011585	0.024116
CISAT	-0.068795	-0.046615	0.039594	0.021676
PWPUMA00	0.043498	0.046923	-0.064990	0.218104
AGE	-0.010589	-0.027549	0.025497	0.037967
CIDIAL	-0.081606	-0.051882	0.038416	0.022467
CIOTHSVC	-0.082161	-0.053583	0.040146	0.023926
PWCOUNTY	0.037754	0.097656	-0.149448	0.264917
HINSEMP	0.147308	-0.002188	-0.057024	-0.002844
MARRNO	-0.093687	-0.009408	-0.087872	-0.013348
HCOVPRIV	0.203457	-0.049415	-0.073043	0.049785
TRANWORK	0.031533	0.043265	-0.002185	-0.007714
DEGFIELD	0.222884	0.293828	0.057056	-0.092124
DEGFIELDD	0.222901	0.293810	0.057043	-0.092101
GCHOUSE	-0.139546	0.089802	0.043450	0.019599
OCC	-0.121307	-0.123197	0.034167	0.002030
POVERTY	0.124621	0.009848	-0.070104	0.015046
YRMARR	-0.088753	0.025907	-0.098506	-0.025626
VETSTAT	-0.047419	-0.059309	0.037960	0.112969

```
TRANTIME   -0.023477 -0.031904  0.012892  0.002316
AVAILBLE   -0.041545 -0.042286  0.072783  0.009687
RIDERS     -0.047638 -0.056095 -0.007016 -0.003394
IND         0.011835  0.014394  0.088465 -0.019963
PWSTATE2   -0.004422  0.035808 -0.124681  0.163798
DEPARTS    -0.015298 -0.060398  0.023471  0.014900
ARRIVES    -0.016075 -0.060142  0.023496  0.014340
CARPOOL    -0.043881 -0.057997 -0.005677 -0.002611
PWTYPE     -0.045167 -0.069557  0.002541 -0.071553
LOOKING    -0.043776 -0.026037  0.044211 -0.011773
OCCSCORE    0.028827  0.045693  0.031780 -0.050337
HWSEI       0.061598  0.076671  0.048008 -0.062362
CLASSWKR   -0.022877 -0.018676  0.056093 -0.030010
UHRSWORK   -0.012952 -0.003863 -0.002151 -0.027314
LABFORCE   -0.036025 -0.038880  0.053398 -0.003279
```

[79]:
```python
fig = px.scatter_matrix(
    df_pandas,
    dimensions=['PC-0','PC-1'],
    color=df_pandas.index
)
fig.update_traces(diagonal_visible=False)
fig.show()
```

[80]:
```python
#save top 5 and bottom 10 loadings so that it gives positive and negative␣
↪loadings
lb0_pos = df_pandas.sort_values('PC-0', ascending=True).head(8).index.str.
↪cat(sep=' ')
lb0_neg = df_pandas.sort_values('PC-0', ascending=False).head(8).index.str.
↪cat(sep=' ')
lb1_pos = df_pandas.sort_values('PC-1', ascending=True).head(8).index.str.
↪cat(sep=' ')
```

```
lb1_neg = df_pandas.sort_values('PC-1', ascending=False).head(8).index.str.
 ↪cat(sep=' ')
```

[81]:
```
#print results
print('PC0+', lb0_pos)
print('PC0-', lb0_neg)
print('PC1+', lb1_pos)
print('PC1-', lb1_neg)
```

```
PC0+ LABFORCE UHRSWORK CLASSWKR HWSEI OCCSCORE LOOKING PWTYPE CARPOOL
PC0- INCTOT MARST HINSCAID HCOVPUB SCHOOL RELATE SCHLTYPE GRADEATT
PC1+ AGE FTOTINC GQ EMPSTAT VETSTAT AVAILBLE HINSCARE GCHOUSE
PC1- MULTGEN FRIDGE BIRTHYR CIDIAL CIOTHSVC CISAT INCTOT NMOTHERS
```

[ ]:
```
!jupyter nbconvert ed_squad_final_smn.ipynb --to pdf
```

[ ]: