In [1]:
```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import multilabel_confusion_matrix
from sklearn import metrics

#pd.set_option('display.max_rows', 100)
```

In [2]:
```
data = pd.read_csv('PIAAC_data.csv')
```

Data Exploration

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 664 entries, 0 to 663
Columns: 588 entries, CNTRYID to PS_class
dtypes: float64(18), int64(394), object(176)
memory usage: 3.0+ MB
```

In [4]:
```
seq_cols = []
for col in data.columns:
    if 'seq' in col:
            seq_cols.append(col)

df_num = data.drop(columns = seq_cols)
```

In [5]:
```
df_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 664 entries, 0 to 663
Columns: 566 entries, CNTRYID to PS_class
dtypes: float64(18), int64(394), object(154)
memory usage: 2.9+ MB
```

In [6]:
```
df_num = df_num.drop(columns = ['CNTRYID', 'SEQID', 'D_Q16b_T', 'ZZ5', 'ZZ6', 'REG_TL2',
```

In [7]:
```
df_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 664 entries, 0 to 663
Columns: 558 entries, U01a000A to PS_class
dtypes: float64(18), int64(393), object(147)
memory usage: 2.8+ MB
```

In [8]:
```
df = df_num.apply(pd.to_numeric, errors='coerce')
```

In [9]:
```
na_cols = df.columns[df.isnull().any()].tolist()
```

In [10]:
```
# Splitting Test/Train
train_data, test_data = train_test_split(df, test_size=0.2, random_state=123)

print(f"No. of training examples: {train_data.shape[0]}")
print(f"No. of testing examples: {test_data.shape[0]}")
```

```
No. of training examples: 531
No. of testing examples: 133
```

In [11]:
```
train_data.info()
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 531 entries, 378 to 510
Columns: 558 entries, U01a000A to PS_class
dtypes: float64(165), int64(393)
memory usage: 2.3 MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 133 entries, 348 to 66
Columns: 558 entries, U01a000A to PS_class
dtypes: float64(165), int64(393)
memory usage: 580.8 KB
```

MISSING VALUES

If scale, take mean. If nominal (catagorical), take mode.

In [12]:
```
var_descr = pd.read_excel('Variable Information.xlsx')
```

In [13]:
```
var_descr.head()
```

Out[13]:

	Variable	Position	Label	Measurement Level
0	CNTRYID	1.0	Country ID	Nominal
1	SEQID	2.0	Sequential ID (randomly derived)	Nominal
2	U01a000A	3.0	Problem-solving Unit 01a (Number of Actions)	Scale
3	U01b000A	4.0	Problem-solving Unit 01b (Number of Actions)	Scale
4	U03a000A	5.0	Problem-solving Unit 03a (Number of Actions)	Scale

In [14]:
```
na_vars_info = var_descr[var_descr['Variable'].isin(na_cols)]
na_vars_info = na_vars_info[['Variable', 'Measurement Level']]
```

In [15]:
```
scale_vars = na_vars_info[na_vars_info['Measurement Level'] == 'Scale']
scale_vars = list(scale_vars['Variable'])
```

In [16]:
```
nom_vars = na_vars_info[na_vars_info['Measurement Level'] == 'Nominal']
nom_vars = list(nom_vars['Variable'])
```

In [17]:
```
ord_vars = na_vars_info[na_vars_info['Measurement Level'] == 'Ordinal']
ord_vars = list(ord_vars['Variable'])
```

In [18]:
```
# Train Data NAs
for col in train_data:
    if col in scale_vars:
        train_data[col] = train_data[col].fillna(train_data[col].mean())
    if col in nom_vars:
        train_data[col] = train_data[col].fillna(train_data[col].mode()[0])
    if col in ord_vars:
        train_data[col] = train_data[col].fillna(train_data[col].median())

train_data.columns[train_data.isnull().any()].tolist()
```

Out[18]:
```
[]
```

In [19]:
```
# Test Data NAs
for col in test_data:
    if col in scale_vars:
        test_data[col] = test_data[col].fillna(test_data[col].mean())
    if col in nom_vars:
        test_data[col] = test_data[col].fillna(test_data[col].mode()[0])
    if col in ord_vars:
        test_data[col] = test_data[col].fillna(test_data[col].median())

test_data.columns[test_data.isnull().any()].tolist()
```

Out[19]:
```
[]
```

In [20]:
```
train_data.info() #558

train_data.to_csv('train_data.csv')
test_data.to_csv('test_data.csv')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 531 entries, 378 to 510
Columns: 558 entries, U01a000A to PS_class
dtypes: float64(165), int64(393)
memory usage: 2.3 MB
```

Random Forest

In [21]:
```
# Spliting data into test and train sets
train_x, train_y = train_data.drop(['PS_class'], axis=1), train_data[['PS_class']]
test_x, test_y = test_data.drop(['PS_class'], axis=1), test_data[['PS_class']]

# fitting the model
model = RandomForestClassifier(n_estimators=42, n_jobs=-1, random_state=123)
model.fit(train_x, train_y.values.ravel())

# Use the forest's predict method on the test data
predictions = model.predict(test_x)
predictions=predictions.reshape(133,1)

# Calculate the absolute errors
errors = abs(predictions - test_y)

# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 3), 'degrees.')
print("Accuracy:",metrics.accuracy_score(test_y, predictions))
print("Precision: ", metrics.precision_score(test_y, predictions, average='micro'))
```

```
Mean Absolute Error: PS_class    0.278
dtype: float64 degrees.
Accuracy: 0.7218045112781954
Precision:  0.7218045112781954
```

```
/Users/cansufreeman/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:
3438: FutureWarning: In a future version, DataFrame.mean(axis=None) will return a scalar
mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or
just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

In [22]:
```
print(metrics.classification_report(test_y, predictions))
```

```
              precision    recall  f1-score   support

           0       0.75      0.68      0.71        31
           1       0.65      0.68      0.67        47
           2       0.77      0.86      0.81        50
           3       0.00      0.00      0.00         5
```

```
    accuracy                           0.72       133
   macro avg       0.54      0.55      0.55       133
weighted avg       0.69      0.72      0.71       133
```

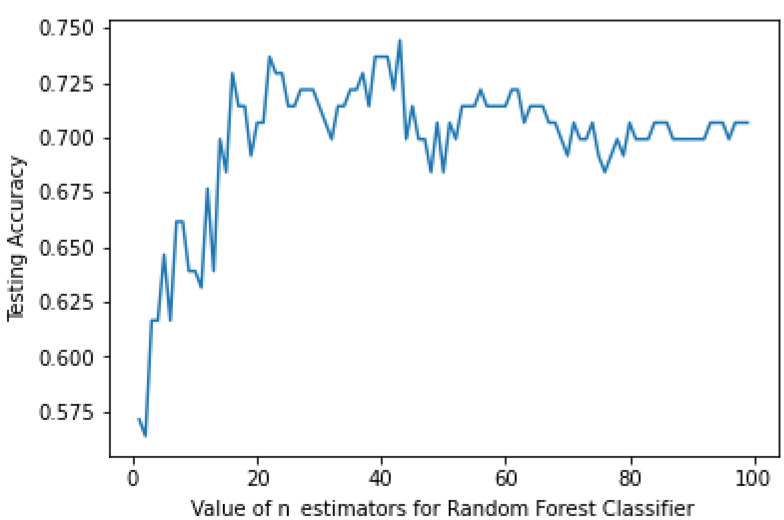
```
/Users/cansufreeman/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classific
ation.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/cansufreeman/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classific
ation.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/cansufreeman/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classific
ation.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [38]:

```python
scores =[]
for k in range(1, 200):
    rfc = RandomForestClassifier(n_estimators=k, n_jobs=-1, random_state=123)
    rfc.fit(train_x, train_y.values.ravel())
    y_pred = rfc.predict(test_x)
    scores.append(metrics.accuracy_score(test_y, y_pred))

import matplotlib.pyplot as plt
%matplotlib inline

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(range(1, 200), scores)
plt.xlabel('Value of n_estimators for Random Forest Classifier')
plt.ylabel('Testing Accuracy')

#plt.savefig('n-estimators')
scores.index(max(scores)), max(scores)
```

Out[38]:

```
(42, 0.7443609022556391)
```



In [23]:

```python
multilabel_confusion_matrix(test_y, predictions)
# TP, FP, TN, FN
```

Out[23]:

```
array([[[ 95,   7],
        [ 10,  21]],
```

```
       [[ 69,  17],
        [ 15,  32]],

       [[ 70,  13],
        [  7,  43]],

       [[128,   0],
        [  5,   0]]])
```

In [ ]:
```
array([[[ 96,   6],
        [  7,  24]],

       [[ 69,  17],
        [ 13,  34]],

       [[ 71,  12],
        [ 10,  40]],

       [[128,   0],
        [  5,   0]]])
```

Fine tuning important variables

In [23]:
```
U_vars = ['U01a000A', 'U01b000A', 'U03a000A', 'U06a000A', 'U06b000A',
          'U21x000A', 'U04a000A', 'U19a000A', 'U19b000A', 'U07x000A',
          'U02x000A', 'U16x000A', 'U11b000A', 'U23x000A']
train_x['U_sum'] = train_x[U_vars].sum(axis=1)
test_x['U_sum'] = test_x[U_vars].sum(axis=1)
```

In [24]:
```
first_interaction_vars = []
time_on_task_vars = []
page_visits_vars = []
diff_emails_vars = []
switch_env_vars = []
num_email_views_vars = []
diff_pages_visit_vars = []
page_revisits_vars = []
revisited_emails_vars = []

for col in train_x:
    if 'first_interaction' in col:
        first_interaction_vars.append(col)
    if 'time_on_task' in col:
        time_on_task_vars.append(col)
    if 'page_visits' in col:
        page_visits_vars.append(col)
    if 'diff_emails' in col:
        diff_emails_vars.append(col)
    if 'environment' in col:
        switch_env_vars.append(col)
    if 'email_views' in col:
        num_email_views_vars.append(col)
    if 'diff_visited' in col:
        diff_pages_visit_vars.append(col)
    if 'revisits' in col:
        page_revisits_vars.append(col)
    if 'revisited' in col:
        revisited_emails_vars.append(col)


train_x['first_interaction'] = train_x[first_interaction_vars].sum(axis=1)
test_x['first_interaction'] = test_x[first_interaction_vars].sum(axis=1)
```

```
train_x['time_on_task'] = train_x[time_on_task_vars].sum(axis=1)
test_x['time_on_task'] = test_x[time_on_task_vars].sum(axis=1)

train_x['page_visits'] = train_x[page_visits_vars].sum(axis=1)
test_x['page_visits'] = test_x[page_visits_vars].sum(axis=1)

train_x['diff_emails'] = train_x[diff_emails_vars].sum(axis=1)
test_x['diff_emails'] = test_x[diff_emails_vars].sum(axis=1)

train_x['switch_env'] = train_x[switch_env_vars].sum(axis=1)
test_x['switch_env'] = test_x[switch_env_vars].sum(axis=1)

train_x['num_email_views'] = train_x[num_email_views_vars].sum(axis=1)
test_x['num_email_views'] = test_x[num_email_views_vars].sum(axis=1)

train_x['diff_pages_visits'] = train_x[diff_pages_visit_vars].sum(axis=1)
test_x['diff_pages_visits'] = test_x[diff_pages_visit_vars].sum(axis=1)

train_x['page_revisits'] = train_x[page_revisits_vars].sum(axis=1)
test_x['page_revisits'] = test_x[page_revisits_vars].sum(axis=1)

train_x['revisited_emails'] = train_x[revisited_emails_vars].sum(axis=1)
test_x['revisited_emails'] = test_x[revisited_emails_vars].sum(axis=1)
```

In [25]:
```
drop_vars = (U_vars + first_interaction_vars + time_on_task_vars + page_visits_vars
             + diff_emails_vars + switch_env_vars +  num_email_views_vars
             + diff_pages_visit_vars + page_revisits_vars + revisited_emails_vars)

created_vars = ['U_sum', 'first_interaction','time_on_task',
                'page_visits', 'diff_emails', 'switch_env', 'num_email_views',
                'diff_pages_visits', 'page_revisits', 'revisited_emails']
```

In [26]:
```
train_x = train_x.drop(columns = drop_vars)
test_x = test_x.drop(columns = drop_vars)
```

In [27]:
```
train_x.to_csv('train_x.csv')
test_x.to_csv('test_x.csv')
train_y.to_csv('train_y.csv')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 531 entries, 378 to 510
Columns: 486 entries, PS1_u01a_num_cancel to revisited_emails
dtypes: float64(96), int64(390)
memory usage: 2.0 MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 133 entries, 348 to 66
Columns: 486 entries, PS1_u01a_num_cancel to revisited_emails
dtypes: float64(96), int64(390)
memory usage: 506.0 KB
```

In [28]:
```
# fitting the model
model = RandomForestClassifier(n_estimators=142, n_jobs=-1, random_state=123)
model.fit(train_x, train_y.values.ravel())

# Use the forest's predict method on the test data
predictions = model.predict(test_x)
predictions=predictions.reshape(133,1)

# Calculate the absolute errors
errors = abs(predictions - test_y)

# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 3), 'degrees.')
print("Accuracy:",metrics.accuracy_score(test_y, predictions))
print("Precision: ", metrics.precision_score(test_y, predictions, average='micro'))
```

```
Mean Absolute Error: PS_class    0.256
dtype: float64 degrees.
Accuracy: 0.7443609022556391
Precision:  0.7443609022556391
```

```
/Users/cansufreeman/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:
3438: FutureWarning: In a future version, DataFrame.mean(axis=None) will return a scalar
mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or
just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

In [32]:
```
scores =[]
for k in range(1, 200):
    rfc = RandomForestClassifier(n_estimators=k, n_jobs=-1, random_state=123)
    rfc.fit(train_x, train_y.values.ravel())
    y_pred = rfc.predict(test_x)
    scores.append(metrics.accuracy_score(test_y, y_pred))

import matplotlib.pyplot as plt
%matplotlib inline

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
#plt.plot(range(1, 200), scores)
#plt.xlabel('Value of n_estimators for Random Forest Classifier')
#plt.ylabel('Testing Accuracy')

#plt.savefig('n-estimators')
scores.index(max(scores)), max(scores)
```

Out[32]:
```
(41, 0.7443609022556391)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [29]:
```
# plotting feature importances from original RF Model with ALL variables
features = train_x.columns
importances = model.feature_importances_
indices = np.argsort(importances)
```

In [30]:
```
important_features = pd.DataFrame(data = {'col':[features[i] for i in indices],
                                          'impt':importances[indices] })

# keep features that are above 1% importance
keep_features = important_features.loc[(important_features[['impt']] >= 0.0075).all(axis
keepf = keep_features['col'].unique()
keepf = keepf.tolist()

# drop features:
drop_features = important_features.loc[(important_features[['impt']] == 0.0).all(axis=1)
dropf = drop_features['col'].unique()
dropf = dropf.tolist()
```

In [31]:
```
keepf
```

Out[31]:
```
['READHOME',
 'ICTHOME',
 'NUMHOME',
```

```
 'first_interaction',
 'time_on_task',
 'revisited_emails',
 'diff_emails',
 'num_email_views',
 'page_visits',
 'switch_env',
 'page_revisits',
 'diff_pages_visits',
 'U_sum',
 'PVLIT1',
 'PVNUM1']
```

In [32]:
```
keepf.append('C_Q09_C')
```

In [33]:
```
train_x = train_x[keepf]
test_x = test_x[keepf]
```

In [39]:
```
# fitting the model
model = RandomForestClassifier(n_estimators=63, n_jobs=-1, random_state=123)
model.fit(train_x, train_y.values.ravel())

# Use the forest's predict method on the test data
predictions = model.predict(test_x)
predictions=predictions.reshape(133,1)

# Calculate the absolute errors
errors = abs(predictions - test_y)

# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 3), 'degrees.')
print("Accuracy:",metrics.accuracy_score(test_y, predictions))
print("Precision: ", metrics.precision_score(test_y, predictions, average='micro'))
```

```
Mean Absolute Error: PS_class    0.263
dtype: float64 degrees.
Accuracy: 0.7368421052631579
Precision:  0.7368421052631579
```

```
/Users/cansufreeman/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:
3438: FutureWarning: In a future version, DataFrame.mean(axis=None) will return a scalar
mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or
just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

In [35]:
```
print(metrics.classification_report(test_y, predictions))
```

```
              precision    recall  f1-score   support

           0       0.80      0.77      0.79        31
           1       0.63      0.70      0.67        47
           2       0.74      0.74      0.74        50
           3       0.00      0.00      0.00         5

    accuracy                           0.71       133
   macro avg       0.54      0.55      0.55       133
weighted avg       0.69      0.71      0.70       133
```

In [45]:
```
multilabel_confusion_matrix(test_y, predictions)
# TP, FP, TN, FN
```

Out[45]:
```
array([[[ 96,   6],
        [  7,  24]],

       [[ 69,  17],
```

```
        [ 13,  34]],

       [[ 71,  12],
        [ 10,  40]],

       [[128,   0],
        [  5,   0]]])
```

In [67]:
```
#all keepf(.0075 + Cvar) = 72.93, 49 n
# rm ['READHOME','NUMHOME', 'PVNUM1', revisited email] 73.68, 63 n
# rm
```

In [37]:
```
scores =[]
for k in range(1, 100):
    rfc = RandomForestClassifier(n_estimators=k, n_jobs=-1, random_state=123)
    rfc.fit(train_x, train_y.values.ravel())
    y_pred = rfc.predict(test_x)
    scores.append(metrics.accuracy_score(test_y, y_pred))

import matplotlib.pyplot as plt
%matplotlib inline

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
#plt.plot(range(1, 100), scores)
#plt.xlabel('Value of n_estimators for Random Forest Classifier')
#plt.ylabel('Testing Accuracy')

#plt.savefig('n-estimators')
scores.index(max(scores)), max(scores)
```

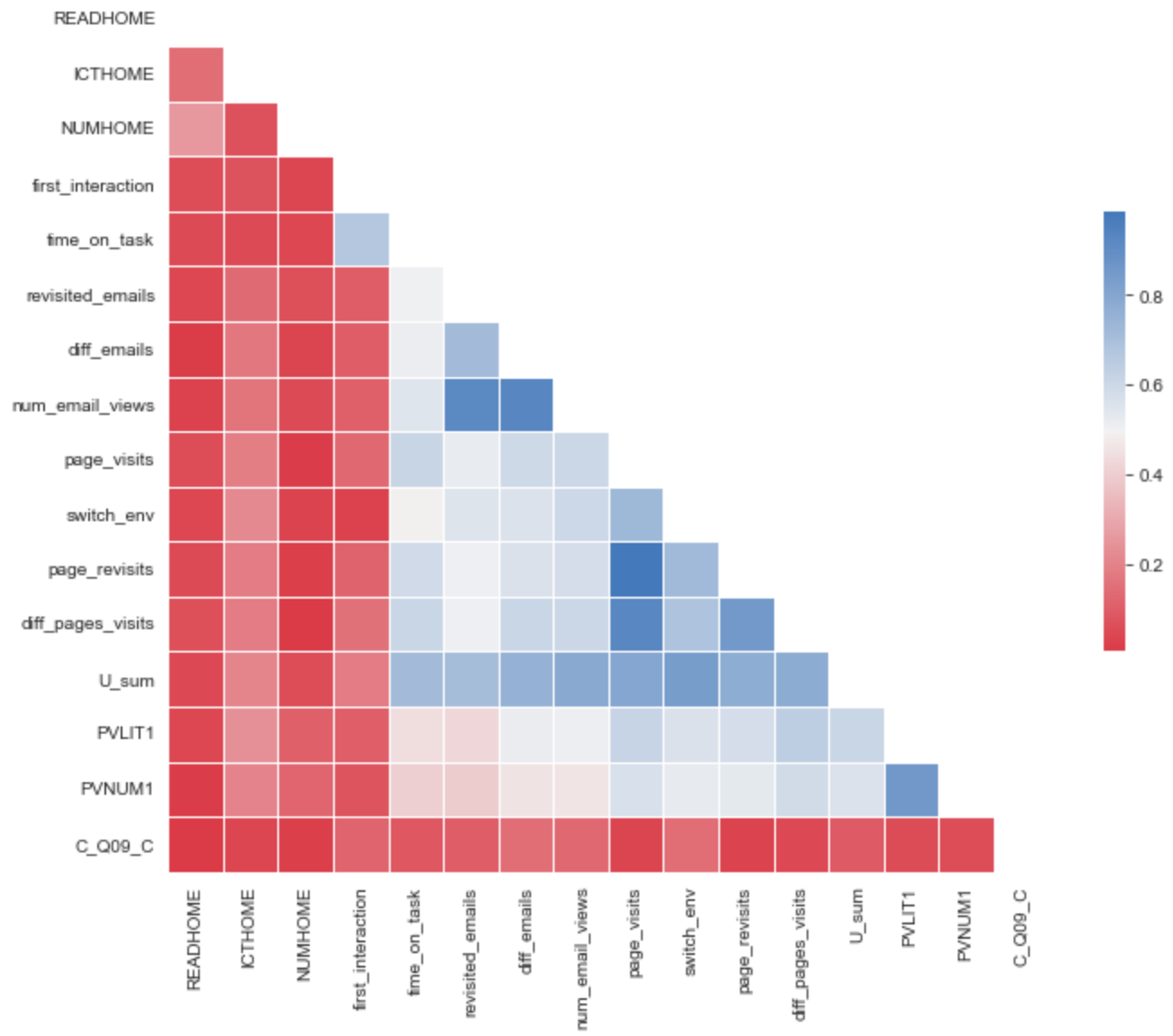
Out[37]:
```
(82, 0.7293233082706767)
```

Check for correlations

In [60]:
```
# use the pands .corr() function to compute pairwise correlations for the dataframe
corr = train_x.corr().abs()

# visualise the data with seaborn
mask = np.triu(np.ones_like(corr, dtype=np.bool))
sns.set_style(style = 'white')

f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(10, 250, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap,
        square=True,
        linewidths=.5, cbar_kws={"shrink": .5}, ax=ax)
#plt.savefig('correlation6.png')
```

```
/var/folders/qk/0w6s_3jd78s0prl6n7n20kd00000gn/T/ipykernel_1596/4004825762.py:5: Deprecation
Warning: `np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  mask = np.triu(np.ones_like(corr, dtype=np.bool))
```

Out[60]:
```
<AxesSubplot:>
```



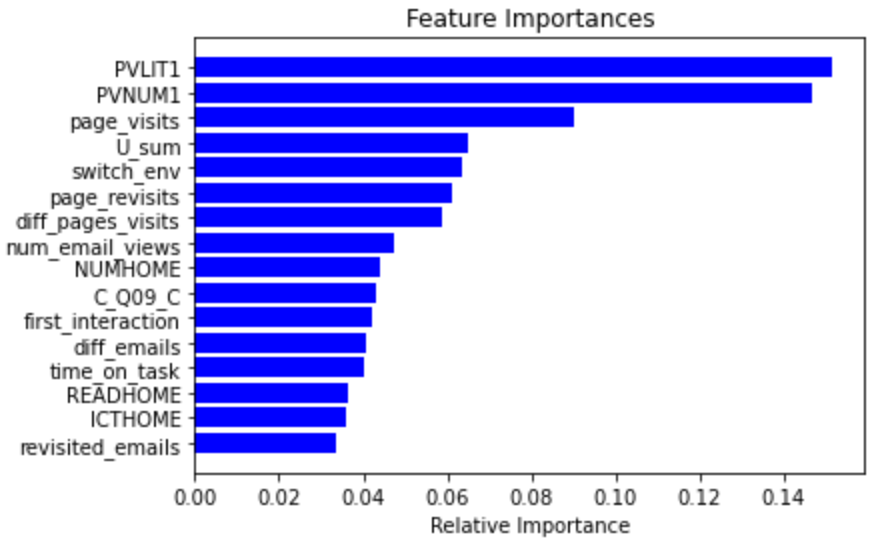
In [61]:
```
train_x.corr().abs()
```

Out[61]:

	READHOME	ICTHOME	NUMHOME	first_interaction	time_on_task	revisited_emails	diff
READHOME	1.000000	0.140528	0.254386	0.056193	0.046346	0.040822	0
ICTHOME	0.140528	1.000000	0.068600	0.071926	0.049046	0.131446	0
NUMHOME	0.254386	0.068600	1.000000	0.035390	0.040855	0.059439	0
first_interaction	0.056193	0.071926	0.035390	1.000000	0.669407	0.096504	0
time_on_task	0.046346	0.049046	0.040855	0.669407	1.000000	0.501389	0
revisited_emails	0.040822	0.131446	0.059439	0.096504	0.501389	1.000000	0
diff_emails	0.009410	0.166061	0.031024	0.096072	0.510933	0.716336	1
num_email_views	0.026507	0.161223	0.048277	0.103922	0.546516	0.920837	
page_visits	0.054859	0.187180	0.015195	0.129583	0.612821	0.521811	0
switch_env	0.039720	0.215621	0.030625	0.026059	0.490922	0.552925	0
page_revisits	0.048956	0.180657	0.018132	0.114681	0.587859	0.505743	0
diff_pages_visits	0.063121	0.183251	0.006700	0.151360	0.608464	0.505869	0
U_sum	0.045381	0.206410	0.056553	0.181708	0.714938	0.709447	0
PVLIT1	0.040230	0.236886	0.107300	0.101327	0.441964	0.423721	

PVNUM1	0.013349	0.201617	0.119687	0.075773	0.401027	0.394906	C
C_Q09_C	0.003844	0.035056	0.017223	0.115899	0.080682	0.099600	

In [34]:
```
# plotting feature importances from original RF Model with ALL variables
features = train_x.columns
importances = model.feature_importances_
indices = np.argsort(importances)

# Plot
plt.figure() #(figsize=(20,50))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.savefig('featureimportances.png')
plt.show()
```



In [ ]:

In [37]:
```
train_x = train_x.drop(columns = ['READHOME', 'PVNUM1', 'NUMHOME', 'revisited_emails'])
test_x = test_x.drop(columns = ['READHOME', 'PVNUM1', 'NUMHOME', 'revisited_emails'])
```

In [65]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [41]:
```
# known from previous analysis
keepf = ['C_Q09_C', 'NUMHOME', 'ICTHOME', 'U01a000A', 'U01b000A', 'PVLIT1']

# Spliting keep data into test and train sets
train_x, train_y = train_data[keepf], train_data[['PS_class']]
test_x, test_y = test_data[keepf], test_data[['PS_class']]
```

In [42]:
```
# fitting the model
model = RandomForestClassifier(n_estimators=10, n_jobs=-1, random_state=123)
model.fit(train_x, train_y.values.ravel())
```

```
# Use the forest's predict method on the test data
predictions = model.predict(test_x)
predictions=predictions.reshape(133,1)

# Calculate the absolute errors
errors = abs(predictions - test_y)

# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 3), 'degrees.')
print("Accuracy:",metrics.accuracy_score(test_y, predictions))
```

```
Mean Absolute Error: PS_class    0.263
dtype: float64 degrees.
Accuracy: 0.7518796992481203
```

```
/Users/cansufreeman/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:
3438: FutureWarning: In a future version, DataFrame.mean(axis=None) will return a scalar
mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or
just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

In [43]:
```
print("Precision: ", metrics.precision_score(test_y, predictions, average='micro'))
```

```
Precision:  0.7518796992481203
```

In [44]:
```
print(metrics.classification_report(test_y, predictions))
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.85        31
           1       0.73      0.70      0.72        47
           2       0.81      0.76      0.78        50
           3       0.17      0.20      0.18         5

    accuracy                           0.75       133
   macro avg       0.63      0.64      0.63       133
weighted avg       0.76      0.75      0.75       133
```

In [45]:
```
multilabel_confusion_matrix(test_y, predictions)
# TP, FP, TN, FN
```

Out[45]:
```
array([[[ 95,   7],
        [  3,  28]],

       [[ 74,  12],
        [ 14,  33]],

       [[ 74,   9],
        [ 12,  38]],

       [[123,   5],
        [  4,   1]]])
```

In [48]:
```
from sklearn.model_selection import cross_val_score

>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scoresscor
```

In [56]:
```
scores = cross_val_score(model, train_x, train_y.values.ravel(), cv=10)
scores
```

Out[56]:
```
array([0.66666667, 0.60377358, 0.62264151, 0.73584906, 0.66037736,
       0.71698113, 0.58490566, 0.77358491, 0.62264151, 0.66037736])
```

In [53]:
```
np.mean(scores)
```

Out[53]: 0.6647798742138364

In [ ]: