```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
import statsmodels.formula.api as smf
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from google.colab import drive
drive.mount('/content/drive')
os.chdir('/MyDrive/IEOR_142_Final_Project')
```

```
Mounted at /content/drive

----------------------------------------------------------------------
-----
FileNotFoundError                          Traceback (most recent call
last)
<ipython-input-13-bd472d22e81a> in <module>()
     11 from google.colab import drive
     12 drive.mount('/content/drive')
---> 13 os.chdir('/MyDrive/IEOR_142_Final_Project')

FileNotFoundError: [Errno 2] No such file or directory:
'/MyDrive/IEOR_142_Final_Project'
```

```
surveydata_2014 = pd.read_csv('survey_2014.csv')
surveydata_2017 = pd.read_csv('survey_2017.csv', encoding='latin-1')
surveydata_2020 = pd.read_csv('survey_2020.csv')
```

```
surveydata_2017['Is your primary role within your company related to
tech/IT?'].value_counts()
```

```
surveydata_2020.columns.to_list()
```

```
survey2020 = surveydata_2020.drop({"#", "Is your primary role within
your company related to tech/IT?"}, axis = 1)
survey2020 = survey2020.rename({'*Are you self-employed?*':
"self_employed",
 "How many employees does your company or organization have?":
"no_employees",
 "Have you ever sought treatment for a mental health disorder from a
mental health professional?": "treatment",
 "What is your gender?": "Gender",
 "What is your age?": "Age",
 "Is your employer primarily a tech company/organization?":
"tech_company",
 "Do you have a family history of mental illness?": "family_history",
 'If you have a mental health disorder, how often do you feel that it
```

```
interferes with your work *when being treated effectively?*':
"work_interface",
 'Does your employer provide mental health benefits as part of
healthcare coverage?': "benefits",
 "Do you know the options for mental health care available under your
employer-provided health coverage?": "care_options",
 "Has your employer ever formally discussed mental health (for
example, as part of a wellness campaign or other official
communication)?": "wellness_program",
 "Does your employer offer resources to learn more about mental health
disorders and options for seeking help?": "seek_help",
 "Is your anonymity protected if you choose to take advantage of
mental health or substance abuse treatment resources provided by your
employer?": "anonymity",
 "If a mental health issue prompted you to request a medical leave
from work, how easy or difficult would it be to ask for that leave?":
"leave",
 "Would you feel comfortable discussing a mental health issue with
your direct supervisor(s)?": "supervisor",
 "Would you feel comfortable discussing a mental health issue with
your coworkers?": "coworkers",
 "Would you be willing to bring up a physical health issue with a
potential employer in an interview?": "phys_health_interview",
 'Would you bring up your *mental* health with a potential employer in
an interview?': "mental_health_interview",
 'What country do you *live* in?': "Country",
 'What US state or territory do you *live* in?': "state"}, axis = 1)
survey2020 = survey2020[["self_employed", "Country", "treatment",
"Gender", "Age", "tech_company", "family_history",
                        "work_interface", "benefits", "care_options",
"wellness_program", "seek_help", "anonymity", "no_employees",
                        "leave", "supervisor", "coworkers",
"phys_health_interview", "mental_health_interview", "state"]]

survey2020['Timestamp'] = '2020'
survey2020.head(1)

survey2017 = surveydata_2017.drop({"#", "Is your primary role within
your company related to tech/IT?"}, axis = 1)
survey2017 = survey2017.rename({"<strong>Are you
self-employed?</strong>": "self_employed",
 "How many employees does your company or organization have?":
"no_employees",
 "Have you ever sought treatment for a mental health disorder from a
mental health professional?": "treatment",
 "What is your gender?": "Gender",
 "What is your age?": "Age",
 "Is your employer primarily a tech company/organization?":
"tech_company",
 "Do you have a family history of mental illness?": "family_history",
 "If you have a mental health disorder, how often do you feel that it
interferes with your work <strong>when being treated
effectively?</strong>": "work_interface",
 "Does your employer provide mental health benefitsÂ\xa0as part of
healthcare coverage?": "benefits",
 "Do you know the options for mental health care available under your
employer-provided health coverage?": "care_options",
 "Has your employer ever formally discussed mental health (for
example, as part of a wellness campaign or other official
communication)?": "wellness_program",
 "Does your employer offer resources to learn more about mental health
disorders and options for seeking help?": "seek_help",
 "Is your anonymity protected if you choose to take advantage of
mental health or substance abuse treatment resources provided by your
employer?": "anonymity",
 "If a mental health issue prompted you to request a medical leave
from work, how easy or difficult would it be to ask for that leave?":
"leave",
 "Would you feel comfortable discussing a mental health issue with
your direct supervisor(s)?": "supervisor",
 "Would you feel comfortable discussing a mental health issue with
your coworkers?": "coworkers",
 "Would you be willing to bring up a physical health issue with a
potential employer in an interview?": "phys_health_interview",
 "Would you bring up your mental health with a potential employer in
an interview?": "mental_health_interview",
 "What country do you <strong>live</strong> in?": "Country",
 "What US state or territory do you <strong>live</strong> in?":
"state",
 "Submit Date (UTC)": "Timestamp"}, axis = 1)
survey2017 = survey2017[["self_employed", "Country", "treatment",
"Gender", "Age", "tech_company", "family_history",
                        "work_interface", "benefits", "care_options",
"wellness_program", "seek_help", "anonymity", "no_employees",
                        "leave", "supervisor", "coworkers",
"phys_health_interview", "mental_health_interview", "state",
"Timestamp"]]
survey2017.head(1)

survey2014 = surveydata_2014.drop({"remote_work",
"mental_health_consequence", "phys_health_consequence",
"mental_vs_physical",
                             "obs_consequence", "comments"}, axis = 1)
survey2014.head(1)

# replace some of the values that are different from the dataset from
survey2014
def replace_values(dataset):
  dataset["self_employed"] = dataset["self_employed"].replace({1:
"Yes", 0: "No"})
  dataset["treatment"] = dataset["treatment"].replace({1: "Yes", 0:
```

```
"No"})
  dataset["tech_company"] = dataset["tech_company"].replace({1: "Yes",
0: "No"})
  dataset["wellness_program"] = dataset["wellness_program"].replace("I
don't know", "Don't know")
  dataset["seek_help"] = dataset["seek_help"].replace("I don't know",
"Don't know")
  dataset["anonymity"] = dataset["anonymity"].replace("I don't know",
"Don't know")
  dataset["leave"] = dataset["leave"].replace("I don't know", "Don't
know")
  dataset["supervisor"] = dataset["supervisor"].replace("Maybe", "Some
of them")
  dataset["coworkers"] = dataset["coworkers"].replace("Maybe", "Some
of them")
  return dataset

survey2017 = replace_values(survey2017)
survey2020 = replace_values(survey2020)

# create a combined dataset
time_list = np.concatenate((survey2014[["Timestamp"]],
survey2017[["Timestamp"]], survey2020[['Timestamp']]), axis = None)
age_list = np.concatenate((survey2014[["Age"]], survey2017[["Age"]],
survey2020[['Age']]), axis = None)
gender_list = np.concatenate((survey2014[["Gender"]],
survey2017[["Gender"]], survey2020[["Gender"]]), axis = None)
country_list = np.concatenate((survey2014[["Country"]],
survey2017[["Country"]], survey2020[["Country"]]), axis = None)
state_list = np.concatenate((survey2014[["state"]],
survey2017[["state"]], survey2020[["state"]]), axis = None)
self_employed_list = np.concatenate((survey2014[["self_employed"]],
survey2017[["self_employed"]], survey2020[["self_employed"]]), axis =
None)
family_history_list = np.concatenate((survey2014[["family_history"]],
survey2017[["family_history"]], survey2020[["family_history"]]), axis
= None)
treatment_list = np.concatenate((survey2014[["treatment"]],
survey2017[["treatment"]], survey2020[["treatment"]]), axis = None)
work_interface_list = np.concatenate((survey2014[["work_interfere"]],
survey2017[["work_interface"]], survey2020[["work_interface"]]), axis
= None)
no_employees_list = np.concatenate((survey2014[["no_employees"]],
survey2017[["no_employees"]], survey2020[["no_employees"]]), axis =
None)
tech_company_list = np.concatenate((survey2014[["tech_company"]],
survey2017[["tech_company"]], survey2020[["tech_company"]]), axis =
None)
benefits_list = np.concatenate((survey2014[["benefits"]],
survey2017[["benefits"]], survey2020[["benefits"]]), axis = None)
care_options_list = np.concatenate((survey2014[["care_options"]],
survey2017[["care_options"]], survey2020[["care_options"]]), axis =
None)
wellness_program_list =
np.concatenate((survey2014[["wellness_program"]],
survey2017[["wellness_program"]], survey2020[["wellness_program"]]),
axis = None)
seek_help_list = np.concatenate((survey2014[["seek_help"]],
survey2017[["seek_help"]], survey2020[["seek_help"]]), axis = None)
anonymity_list = np.concatenate((survey2014[["anonymity"]],
survey2017[["anonymity"]], survey2020[["anonymity"]]), axis = None)
leave_list = np.concatenate((survey2014[["leave"]],
survey2017[["leave"]], survey2020[["leave"]]), axis = None)
supervisor_list = np.concatenate((survey2014[["supervisor"]],
survey2017[["supervisor"]], survey2020[["supervisor"]]), axis = None)
coworkers_list = np.concatenate((survey2014[["coworkers"]],
survey2017[["coworkers"]], survey2020[["coworkers"]]), axis = None)
phys_health_interview_list =
np.concatenate((survey2014[["phys_health_interview"]],
survey2017[["phys_health_interview"]],
survey2020[["phys_health_interview"]]), axis = None)
mental_health_interview_list =
np.concatenate((survey2014[["mental_health_interview"]],
survey2017[["mental_health_interview"]],
survey2020[["mental_health_interview"]]), axis = None)

df_dict = {"Timestamp": time_list, "Age": age_list, "Gender":
gender_list, "Country": country_list,
           "state": state_list, "self_employed": self_employed_list,
"family_history": family_history_list,
           "treatment": treatment_list, "work_interface":
work_interface_list, "no_employees": no_employees_list,
           "tech_company": tech_company_list, "benefits":
benefits_list, "care_options": care_options_list,
           "wellness_program": wellness_program_list, "seek_help":
seek_help_list, "anonymity": anonymity_list,
           "leave": leave_list, "supervisor": supervisor_list,
"coworkers": coworkers_list,
           "phys_health_interview": phys_health_interview_list,
           "mental_health_interview": mental_health_interview_list}

final_df = pd.DataFrame(df_dict)
final_df

                Timestamp   Age  ... phys_health_interview
mental_health_interview
0     2014-08-27 11:29:31  37.0  ...                 Maybe
No
1     2014-08-27 11:29:37  44.0  ...                    No
No
2     2014-08-27 11:29:44  32.0  ...                   Yes
```

```
Yes
3     2014-08-27 11:29:46  31.0  ...                 Maybe
Maybe
4     2014-08-27 11:30:22  31.0  ...                   Yes
Yes
...                   ...   ...  ...                   ...
...
2190                 2020  53.0  ...                    No
No
2191                 2020  23.0  ...                 Maybe
Maybe
2192                 2020  34.0  ...                 Maybe
No
2193                 2020  43.0  ...                    No
No
2194                 2020  37.0  ...                   Yes
Yes

[2195 rows x 21 columns]
```

```
# drop the unnecessary columns
final_df2 = final_df.copy()
final_df2 = final_df2.drop(["Timestamp", "state"], axis = 1)
# final_df2 = final_df2.loc[~final_df2["self_employed"].isna()]
# final_df3 = final_df2.dropna()
# final_df3
final_df2
```

```
       Age  Gender  ... phys_health_interview mental_health_interview
0     37.0  Female  ...                 Maybe                      No
1     44.0       M  ...                    No                      No
2     32.0    Male  ...                   Yes                     Yes
3     31.0    Male  ...                 Maybe                   Maybe
4     31.0    Male  ...                   Yes                     Yes
...    ...     ...  ...                   ...                     ...
2190  53.0    Male  ...                    No                      No
2191  23.0       F  ...                 Maybe                   Maybe
2192  34.0    Male  ...                 Maybe                      No
2193  43.0    Male  ...                    No                      No
2194  37.0    male  ...                   Yes                     Yes

[2195 rows x 19 columns]
```

```
# cleaning the gender column
gender_list = final_df2["Gender"].unique()
male_list = ["Male", "male", "M", "m", "Make", "Male ", "Cis Male",
"Male-ish", "Mal", "msle", "Mail", "cis male",
            "Malr", "Cis Man", 'cis-male', 'male/androgynous ', 'cis
hetero male', 'Male (cis)',
            "male (hey this is the tech industry you're talking
about)", 'God King of the Valajar',
            'Cis male', 'man', 'Male, cis', 'cis male ', 'dude',
'maile', 'Male (CIS)', 'Man', 'Cis-male','cisgender male', 'MAle',
             'male', 'mail']

female_list = ["Female", "female", "Cis Female", "F", "f",
"queer/she/they", "woman", "Female ", "cis-female/femme",
              "Woman", "femail", "Female (cis)", 'Female ', "femalw",
'female (cis)', 'My sex is female.',
              'female (cisgender)', 'Female (cis) ', 'Woman-
identified', 'cis-Female', 'cis female',
              'F, cisgender', 'Female-ish', 'Trans-female', 'Femake',
'FEMALE', 'female, she/her']

other_list = ['Trans-female', 'non-binary', 'Nah', 'All', 'Enby',
'fluid', 'Genderqueer', 'Androgyne', 'Agender',
              'Guy (-ish) ^_^', 'male leaning androgynous', 'Trans
woman', 'Neuter', 'Female (trans)',
              'queer', 'ostensibly male, unsure what that really
means', "Genderfluid", "Nonbinary",
             'Non-binary', 'Agender/genderfluid', 'uhhhhhhhhh fem
genderqueer?', 'Contextual', 'Non binary',
             'Genderqueer demigirl', 'Genderqueer/non-binary',
'nonbinary', 'trans woman', 'Transfeminine',
             'None', 'something kinda male?', 'non-binary', 'trans',
'queer', 'mostly male']

other_list2 = ["p", "A little about you", 'sometimes', '\\-', ]
final_df2["Gender"].replace(male_list, "male", inplace = True)
final_df2["Gender"].replace(female_list, "female", inplace = True)
final_df2["Gender"].replace(other_list, "other", inplace = True)
final_df2 = final_df2.dropna(subset = ["Gender"])
final_df3 = final_df2[~final_df2["Gender"].isin(other_list2)]
final_df3

       Age  Gender  ... phys_health_interview mental_health_interview
0     37.0  female  ...                 Maybe                      No
1     44.0    male  ...                    No                      No
2     32.0    male  ...                   Yes                     Yes
3     31.0    male  ...                 Maybe                   Maybe
4     31.0    male  ...                   Yes                     Yes
...    ...     ...  ...                   ...                     ...
2190  53.0    male  ...                    No                      No
2191  23.0  female  ...                 Maybe                   Maybe
2192  34.0    male  ...                 Maybe                      No
2193  43.0    male  ...                    No                      No
2194  37.0    male  ...                   Yes                     Yes

[2180 rows x 19 columns]
```

```
# check Gender columns
final_df3["Gender"].unique()
```

```
array(['female', 'male', 'other'], dtype=object)
```

```
final_df4 = final_df3.copy()
```

```
labelDict = {}
for feature in final_df4:
    le = preprocessing.LabelEncoder()
    final_df4[feature] = le.fit_transform(final_df4[[feature]])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/
_label.py:115: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
scaler = MinMaxScaler()
final_df4["Age"] = scaler.fit_transform(final_df4[["Age"]])
final_df4
```

```
           Age  Gender  ...  phys_health_interview
mental_health_interview
0     0.431034       0  ...                      0
1
1     0.551724       1  ...                      1
1
2     0.344828       1  ...                      2
2
3     0.327586       1  ...                      0
0
4     0.327586       1  ...                      2
2
...        ...     ...  ...                    ...
...
2190  0.706897       1  ...                      1
1
2191  0.189655       0  ...                      0
0
2192  0.379310       1  ...                      0
1
2193  0.534483       1  ...                      1
1
2194  0.431034       1  ...                      2
2

[2180 rows x 19 columns]
```

```
# X = final_df4[['Age', 'Gender', 'Country', 'self_employed',
'family_history', 'work_interface', 'no_employees',
#                'tech_company', 'benefits', 'care_options',
'wellness_program', 'seek_help', 'anonymity',
#                'leave', 'supervisor', 'coworkers',
'phys_health_interview', 'mental_health_interview']]
# y = final_df4["treatment"]
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.33, random_state = 42)
train, test = train_test_split(final_df4, test_size = 0.1,
random_state = 42)

# Logistic Regression

logreg = smf.logit(formula = "treatment ~ Age + Gender + Country +
self_employed + family_history + work_interface +\
                   no_employees + tech_company + benefits +
care_options + wellness_program + seek_help + \
                   anonymity + leave + supervisor + coworkers +
phys_health_interview + mental_health_interview",
                  data = train).fit()
print(logreg.summary)
```

```
Optimization terminated successfully.
         Current function value: 0.591892
         Iterations 5
<bound method BinaryResults.summary of
<statsmodels.discrete.discrete_model.LogitResults object at
0x7f86628af850>>
```

```
y_test = test['treatment']
y_prob = logreg.predict(test)
y_pred = pd.Series([1 if x > 0.5 else 0 for x in y_prob], index =
y_prob.index)

print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
# The proportion of people that we correctly classified.
```

```
Accuracy: 0.6560
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[58 47]
 [28 85]]
```

```
# LDA
# predicting treatment based on features

X_train = train.drop(columns = ['treatment'])
X_test = test.drop(columns = ['treatment'])

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()
```

```
y_train = train['treatment']
y_test = test['treatment']
lda = lda.fit(X_train, y_train)
lda
```

```
LinearDiscriminantAnalysis()
```

```
y_pred = lda.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix: \n", cm)
#from sklearn.metrics import accuracy_score
#print("Accuracy is: \n", accuracy_score(y_test, y_pred))
acc2 = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
TPR2 = cm.ravel()[3]/(cm.ravel()[3]+cm.ravel()[2])
FPR2 = cm.ravel()[1]/(cm.ravel()[1]+cm.ravel()[0])
print('Accuracy is: %.4f' %acc2)
print('TPR is: %.4f' % TPR2)
print('FPR is: %.4f' % FPR2)
```

```
Confusion Matrix:
 [[58 47]
 [28 85]]
Accuracy is: 0.6560
TPR is: 0.7522
FPR is: 0.4476
```

```
0.6559633027522935
```

```
# CART Model
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
# choose optimal ccp_alpha:
from sklearn.model_selection import GridSearchCV

grid_values = {'ccp_alpha': np.linspace(0, 0.10, 201),
               'min_samples_leaf': [5],
               'min_samples_split': [20],
               'max_depth': [30],
               'class_weight' : ['balanced'],
               'random_state': [88]}
#more grid values -- will try every combo of hyper-parameters

dtc = DecisionTreeClassifier()
dtc_cv_acc = GridSearchCV(dtc, param_grid = grid_values, scoring =
'accuracy', cv=5, verbose=1)
dtc_cv_acc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 201 candidates, totalling 1005 fits
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'ccp_alpha': array([0.    , 0.0005, 0.001 ,
0.0015, 0.002 , 0.0025, 0.003 , 0.0035,
       0.004 , 0.0045, 0.005 , 0.0055, 0.006 , 0.0065, 0.007 , 0.0075,
       0.008 , 0.0085, 0.009 , 0.0095, 0.01  , 0.0105, 0.011 , 0.0115,
       0.012 , 0.0125, 0.013 , 0.0135, 0.014 , 0.0145, 0.015 , 0.0155,
       0.016 , 0.0165, 0.017 , 0.0175, 0.018 , 0.0185, 0.019 , 0.0195,
       0.02  , 0.020...
       0.084 , 0.0845, 0.085 , 0.0855, 0.086 , 0.0865, 0.087 , 0.0875,
       0.088 , 0.0885, 0.089 , 0.0895, 0.09  , 0.0905, 0.091 , 0.0915,
       0.092 , 0.0925, 0.093 , 0.0935, 0.094 , 0.0945, 0.095 , 0.0955,
       0.096 , 0.0965, 0.097 , 0.0975, 0.098 , 0.0985, 0.099 , 0.0995,
       0.1   ]),
                         'class_weight': ['balanced'], 'max_depth':
[30],
                         'min_samples_leaf': [5], 'min_samples_split':
[20],
                         'random_state': [88]},
             scoring='accuracy', verbose=1)
```

```
acc = dtc_cv_acc.cv_results_['mean_test_score'] # what sklearn calls
mean_test_score is the holdout set, i.e. the validation set.
ccp = dtc_cv_acc.cv_results_['param_ccp_alpha'].data
```

```
pd.DataFrame({'ccp alpha' : ccp, 'Validation Accuracy': acc}).head(20)
```

```
   ccp alpha  Validation Accuracy
0          0             0.779318
1     0.0005             0.779318
2      0.001             0.787987
3     0.0015             0.806334
4      0.002             0.814987
5     0.0025             0.819066
6      0.003             0.816521
7     0.0035             0.816521
8      0.004             0.817031
9     0.0045             0.817031
10     0.005             0.812960
11    0.0055             0.814491
12     0.006             0.814491
13    0.0065             0.814491
14     0.007             0.814491
15    0.0075             0.814491
16     0.008             0.814491
17    0.0085             0.812960
18     0.009             0.812960
19    0.0095             0.812960
```

```python
print('Grid best parameter ccp_alpha (max. accuracy): ',
dtc_cv_acc.best_params_['ccp_alpha'])
print('Grid best score (accuracy): ', dtc_cv_acc.best_score_)
```

```
Grid best parameter ccp_alpha (max. accuracy):  0.0025
Grid best score (accuracy):  0.819065794256634
```

```python
dtc_cv_acc = dtc_cv_acc.fit(X_train, y_train)
y_pred_cv_acc = dtc_cv_acc.best_estimator_.predict(X_test)
cm2 = confusion_matrix(y_test, y_pred_cv_acc)
print ("Confusion Matrix : \n", cm2)

acc2 = (cm2.ravel()[0]+cm2.ravel()[3])/sum(cm2.ravel())
TPR2 = cm2.ravel()[3]/(cm2.ravel()[3]+cm2.ravel()[2])
FPR2 = cm2.ravel()[1]/(cm2.ravel()[1]+cm2.ravel()[0])
print('Accuracy is: %.4f' %acc2)
print('TPR is: %.4f' % TPR2)
print('FPR is: %.4f' % FPR2)
```

```
Fitting 5 folds for each of 201 candidates, totalling 1005 fits
Confusion Matrix :
 [[82 23]
 [21 92]]
Accuracy is: 0.7982
TPR is: 0.8142
FPR is: 0.2190
```

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
os.chdir('/content/drive/MyDrive/IEOR 142 Final Project')
```

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

```
os.listdir()
```

```
['Cipriani et al_GRISELDA_Lancet 2018_Open data.xlsx',
 'IEOR142-project-guidelines-Fall2021.pdf',
 'DepressionPredictionData',
 'Project Proposal Group 30.gdoc',
 'combined_survey.csv',
 'combined_survey2.csv',
 'Write Up   Working Doc.gdoc',
 'survey14_17_20.csv',
 'survey14_17_20(2).csv',
 'Final_Project_IEOR142_Analytical_Models.ipynb',
 'tech_company_survey.csv',
 'Presentation.gslides',
 'Presentation Script.gdoc',
 'DataCleaning2.ipynb',
 'final_survey_data.csv',
 'CCP alpha.png',
 'num_features.png',
 'bootstrap_test_accuracy.png',
 'Random_Forest_AUC.png',
 'Visualizations.ipynb',
 'Notebook.ipynb',
 'Report.gdoc',
 'Random_Forest_model.ipynb']
```

```
combined_survey = pd.read_csv('survey14_17_20.csv')
combined_survey2 = pd.read_csv('tech_company_survey.csv')
```

```
combined_survey = combined_survey.drop(['Unnamed: 0'], axis =1)
```

```
np.unique(combined_survey['treatment'].apply(lambda x: str(x)))
```

```
array(['No', 'Yes', 'nan'], dtype=object)
```

```
from sklearn.model_selection import train_test_split
y = combined_survey2['treatment']
X = combined_survey2.drop(['treatment'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=69)
```

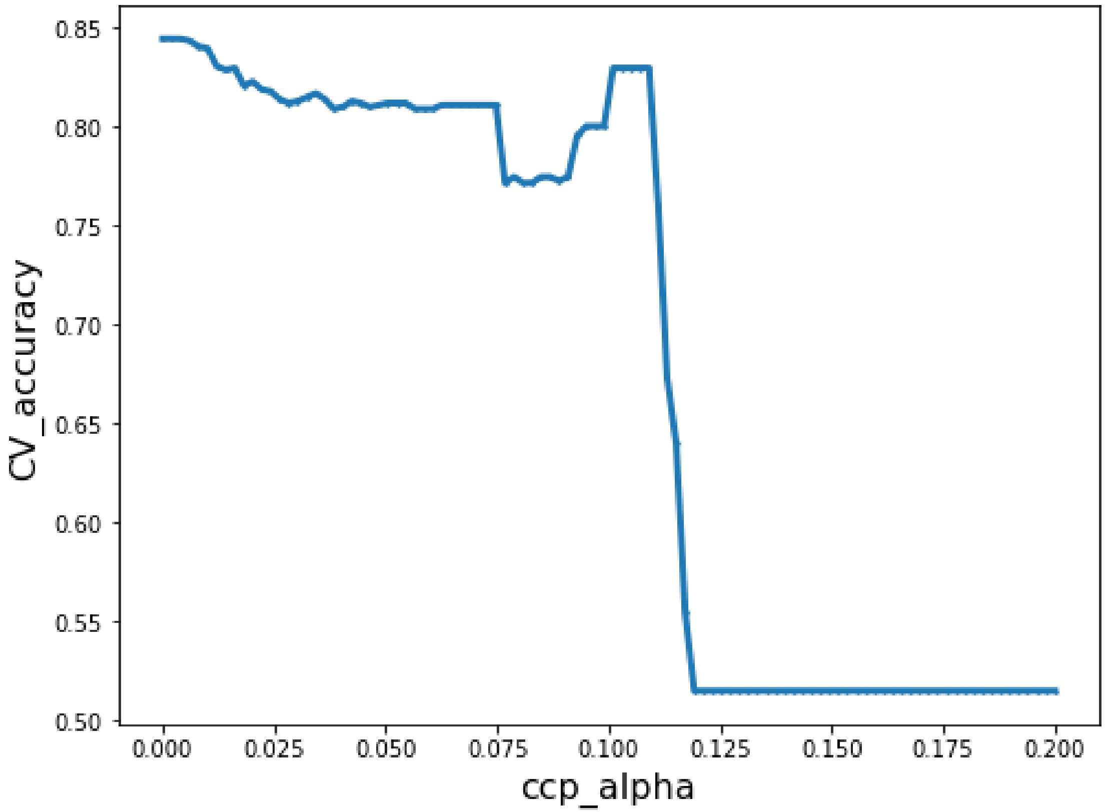
```
X_train.head()
```

```
      Unnamed: 0       Age  ...  phys_health_interview
mental_health_interview
118          382  0.214286  ...                      0
1
1106        1644  0.303571  ...                      0
1
486          911  0.767857  ...                      0
1
512          940  0.303571  ...                      0
1
283          611  0.535714  ...                      0
1

[5 rows x 18 columns]
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
#from sklearn.model import KFold
grid_values = {'ccp_alpha': np.linspace(0,0.2, 100),
              'min_samples_leaf': [5], # min number samples required
for a leaf node
              'min_samples_split': [20], # min number of observations
in a bucket required for a split
              'max_depth': [30],# max nodes of a decision tree
              'random_state': [88]}
rfc = RandomForestClassifier()
#cv = KFold
rfc_cv_acc = GridSearchCV(rfc, param_grid = grid_values, scoring =
'accuracy', cv = 8, verbose = 1)
# note that the default metric used to optimized grid_value is
accuracy
rfc_cv_acc.fit(X_train, y_train)
```

```
Fitting 8 folds for each of 100 candidates, totalling 800 fits

GridSearchCV(cv=8, estimator=RandomForestClassifier(),
             param_grid={'ccp_alpha': array([0.        , 0.0020202 ,
0.0040404 , 0.00606061, 0.00808081,
       0.01010101, 0.01212121, 0.01414141, 0.01616162, 0.01818182,
       0.02020202, 0.02222222, 0.02424242, 0.02626263, 0.02828283,
       0.03030303, 0.03232323, 0.03434343, 0.03636364, 0.03838384,
       0.04040404, 0.04242424, 0.04444444, 0.04646465, 0.04848485,
       0...
       0.15151515, 0.15353535, 0.15555556, 0.15757576, 0.15959596,
       0.16161616, 0.16363636, 0.16565657, 0.16767677, 0.16969697,
       0.17171717, 0.17373737, 0.17575758, 0.17777778, 0.17979798,
       0.18181818, 0.18383838, 0.18585859, 0.18787879, 0.18989899,
       0.19191919, 0.19393939, 0.1959596 , 0.1979798 , 0.2       ]),
                         'max_depth': [30], 'min_samples_leaf': [5],
                         'min_samples_split': [20], 'random_state':
[88]},
             scoring='accuracy', verbose=1)
```

```
import matplotlib.pyplot as plt
ccp_alpha = rfc_cv_acc.cv_results_['param_ccp_alpha'].data
r2_score = rfc_cv_acc.cv_results_['mean_test_score']

plt.figure(figsize = (8,6))
plt.xlabel('ccp_alpha', fontsize = 16)
plt.ylabel('CV_accuracy', fontsize = 16)
plt.scatter(ccp_alpha, r2_score, s = 2)
plt.plot(ccp_alpha, r2_score, linewidth = 3)
plt.savefig('CCP alpha', dpi = 600)
```



```
print('Best CCP_alpha:', rfc_cv_acc.best_params_)
print('Grid best score Accuracy:', rfc_cv_acc.best_score_)
```

```
Best CCP_alpha: {'ccp_alpha': 0.0, 'max_depth': 30,
'min_samples_leaf': 5, 'min_samples_split': 20, 'random_state': 88}
Grid best score Accuracy: 0.8440037524606299
```

```
rfc1 = RandomForestClassifier()
rfc1.fit(X_train, y_train)
pd.DataFrame({'feature': X_train.columns, 'Importance Score':
100*rfc1.feature_importances_})
```

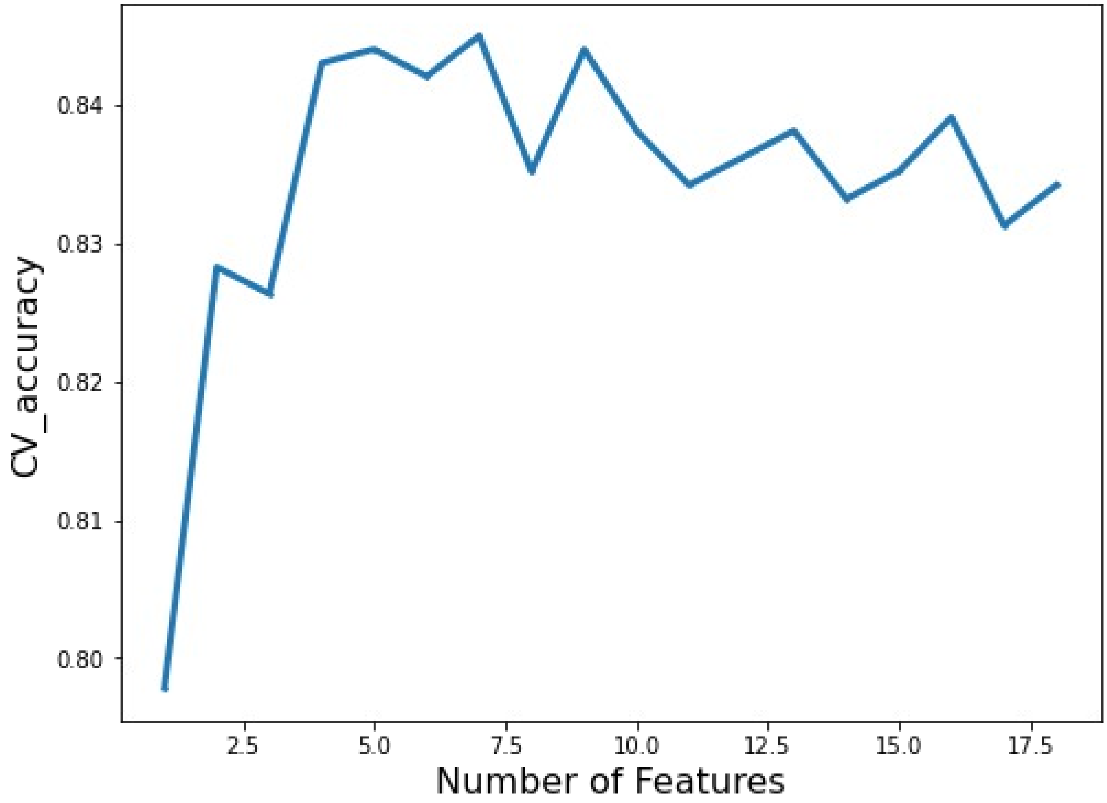
```
                    feature  Importance Score
0                Unnamed: 0          9.487269
1                       Age          8.218859
2                    Gender          2.230060
3                   Country          6.612917
4             self_employed          1.027684
5            family_history          9.505047
6            work_interface         30.253568
7              no_employees          4.204577
8                  benefits          3.213617
9              care_options          4.761176
10         wellness_program          1.906289
11                seek_help          2.374829
12                anonymity          1.845887
13                    leave          3.994765
14               supervisor          2.806414
15                coworkers          3.213400
16    phys_health_interview          2.505384
17  mental_health_interview          1.838258
```

```
# cross validate number of columns
param_grid2 = {
    'max_features': list(range(1, len(X_train.columns) + 1)),
    'max_depth': [50],
    'criterion': ['entropy']}
rf2 = RandomForestClassifier()
rf_cv_acc2 = GridSearchCV(rf2,param_grid = param_grid2, scoring =
'accuracy', cv = 8, verbose = 1 )
rf_cv_acc2.fit(X_train, y_train)
```

```
Fitting 8 folds for each of 18 candidates, totalling 144 fits
```

```
GridSearchCV(cv=8, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['entropy'], 'max_depth': [50],
                         'max_features': [1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12,
                                          13, 14, 15, 16, 17, 18]},
             scoring='accuracy', verbose=1)
```

```
ccp_alpha = rf_cv_acc2.cv_results_['param_max_features'].data
r2_score = rf_cv_acc2.cv_results_['mean_test_score']

plt.figure(figsize = (8,6))
plt.xlabel('Number of Features', fontsize = 16)
plt.ylabel('CV_accuracy', fontsize = 16)
plt.scatter(ccp_alpha, r2_score, s = 2)
plt.plot(ccp_alpha, r2_score, linewidth = 3)
plt.savefig('num_features', dpi = 600)
```



```
print('Grid best Max Features:',
rf_cv_acc2.best_params_['max_features'])
```

```
Grid best Max Features: 7
```

```
final_rfc = RandomForestClassifier(max_features=
rf_cv_acc2.best_params_['max_features'],
ccp_alpha=rfc_cv_acc.best_params_['ccp_alpha'])
final_rfc.fit(X_train, y_train)
final_rfc_pred = final_rfc.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
accuracy_score_1 = accuracy_score(y_test, final_rfc_pred)
print('Random Forest Classification Model', accuracy_score_1)
```

```
Random Forest Classification Model 0.7833001988071571
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, final_rfc_pred)
print('Confusion Matrix: \n', cm)
```

```
Confusion Matrix:
 [[172  57]
 [ 52 222]]
```

```
y_test
```

```
47      1
830     0
441     0
1323    0
213     1
       ..
1123    0
878     0
1265    0
650     0
1109    1
Name: treatment, Length: 503, dtype: int64
```

```
tpr_rate = cm[1][1]/(cm[1][1] + cm[0][1])
fpr_rate = cm[1][0]/(cm[1][0]+ cm[0][0])
```

```
print('True Positive Rate:' , tpr_rate)
print('False Positive Rate:', fpr_rate)
```

```
True Positive Rate: 0.7956989247311828
False Positive Rate: 0.23214285714285715
```

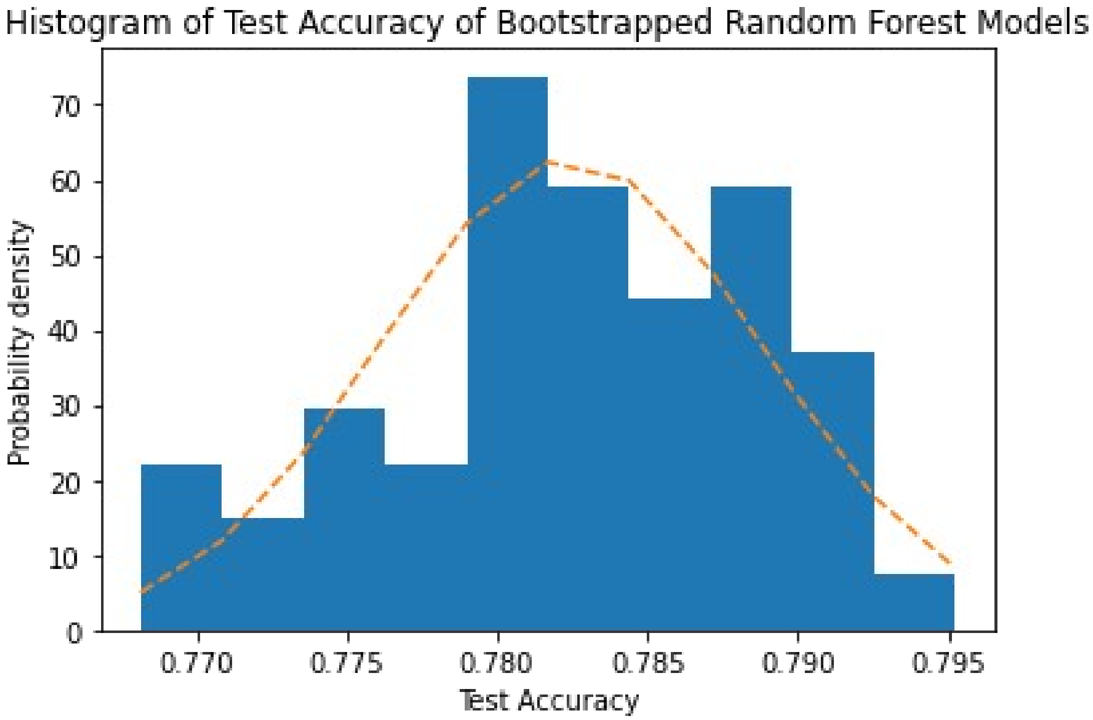
```
from sklearn.metrics import accuracy_score
def bootstrap_validation(sample_num, model,  X_traindata, X_testdata,
y_traindata, y_testdata):
  model = model.fit(X_traindata, y_traindata)
  output_array = []
  for bs_iter in range(sample_num):
    bs_index = np.random.choice(X_testdata.index, len(X_testdata),
replace = True)
    bs_data = X_testdata.loc[bs_index]
    bs_label = y_testdata.loc[bs_index]
    bs_predictions = model.predict(bs_data)
    output_array.append(accuracy_score(bs_label, bs_predictions))

  return pd.DataFrame(output_array)
```

```
rfc = RandomForestClassifier(max_features=
rf_cv_acc2.best_params_['max_features'],
ccp_alpha=rfc_cv_acc.best_params_['ccp_alpha'])
bootstraps  = bootstrap_validation( 50, rfc,  X_train, X_test,
y_train, y_test)
```

```
CI_bootstrap = np.quantile(bootstraps, np.array([0.25, 0.75]))
CI_bootstrap
```

```
array([0.77335984, 0.79870775])
```

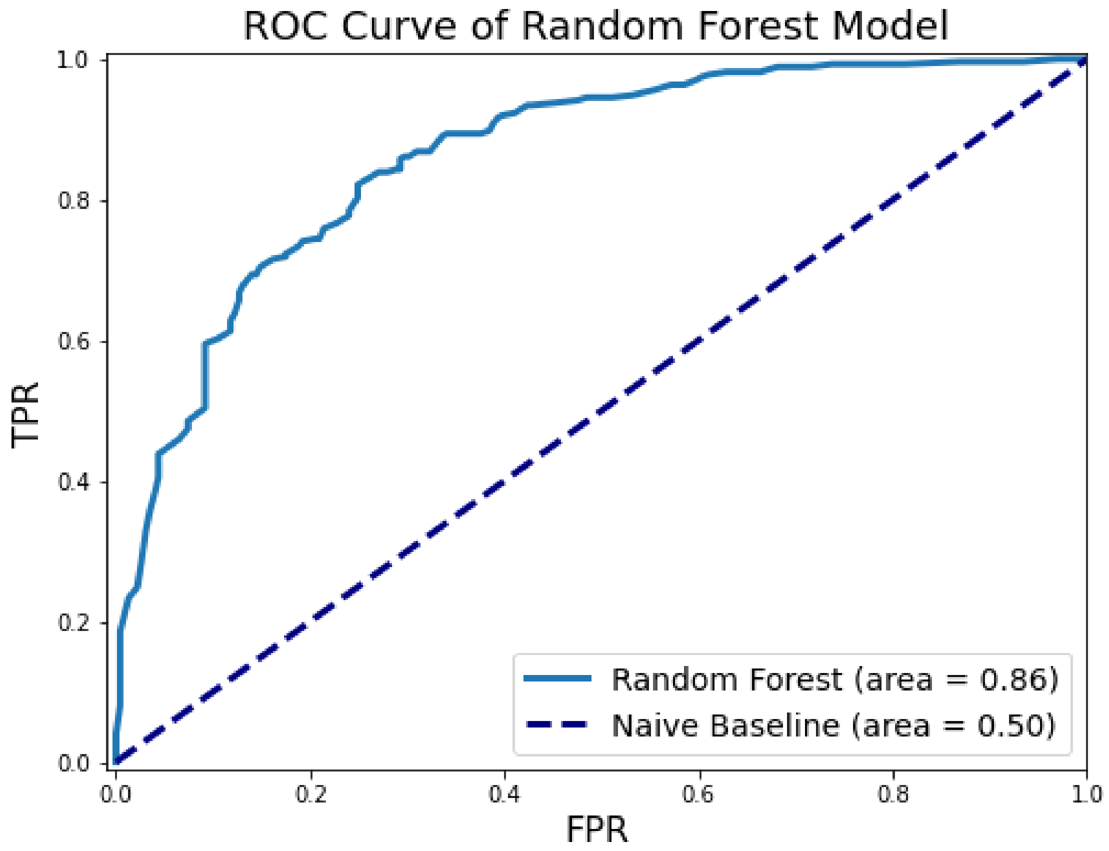
```
CI_list = []
for i in range(0,50):
  bootstraps = bootstrap_validation( 50, rfc,  X_train, X_test,
y_train, y_test)
  CI_bootstraps = np.quantile(bootstraps,np.array([0.25,
0.75])).tolist()
  CI_list.append(CI_bootstraps)

CI_list = np.array(CI_list)
lower = CI_list[:,0]

high_bound = CI_list[:,1]

effect_measure = [sum(x)/len(x) for x in CI_list]

fig, ax = plt.subplots()
num_bins = 10
# the histogram of the data
n, bins, patches = ax.hist(effect_measure, num_bins, density=True)

mu = np.mean(effect_measure)  # mean of distribution
sigma = np.std(effect_measure)  # standard deviation of distribution
# add a 'best fit' line
y = ((1 / (np.sqrt(2 * np.pi) * sigma)) *
     np.exp(-0.5 * (1 / sigma * (bins - mu))**2))
ax.plot(bins, y, '--')
ax.set_xlabel('Test Accuracy')
ax.set_ylabel('Probability density')
ax.set_title('Histogram of Test Accuracy of Bootstrapped Random Forest
Models')

# Tweak spacing to prevent clipping of ylabel
plt.savefig('bootstrap_test_accuracy', dpi = 600)
```



```
bootstraps

            0
0   0.769384
1   0.797217
2   0.781312
3   0.813121
4   0.797217
5   0.757455
6   0.807157
7   0.811133
8   0.763419
9   0.783300
10  0.775348
11  0.787276
12  0.815109
13  0.745527
14  0.801193
15  0.795229
16  0.787276
17  0.777336
18  0.767396
19  0.813121
20  0.823062
21  0.779324
22  0.809145
23  0.785288
24  0.801193
```

```
25  0.759443
26  0.797217
27  0.793241
28  0.795229
29  0.769384
30  0.765408
31  0.821074
32  0.789264
33  0.787276
34  0.795229
35  0.783300
36  0.795229
37  0.737575
38  0.811133
39  0.801193
40  0.799205
41  0.809145
42  0.799205
43  0.821074
44  0.763419
45  0.775348
46  0.803181
47  0.787276
48  0.785288
49  0.793241
```

```
# AUC curve
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

y_prob = final_rfc.predict_proba(X_test)[:,1]

fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.title('ROC Curve of Random Forest Model', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr, tpr, lw=3, label='Random Forest (area =
{:0.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--',
label='Naive Baseline (area = 0.50)')
plt.legend(loc='lower right', fontsize=14)
plt.show()
plt.savefig('Random_Forest_AUC', dpi = 600)
```



```
<Figure size 432x288 with 0 Axes>
```

```
plt.savefig('Random_Forest_AUC', dpi = 600)
```

```
<Figure size 432x288 with 0 Axes>
```

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
os.chdir('/content/drive/MyDrive/IEOR 142 Final Project')
```

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

```
combined_survey2 = pd.read_csv('tech_company_survey.csv')
combined_survey = pd.read_csv('survey14_17_20.csv')
```

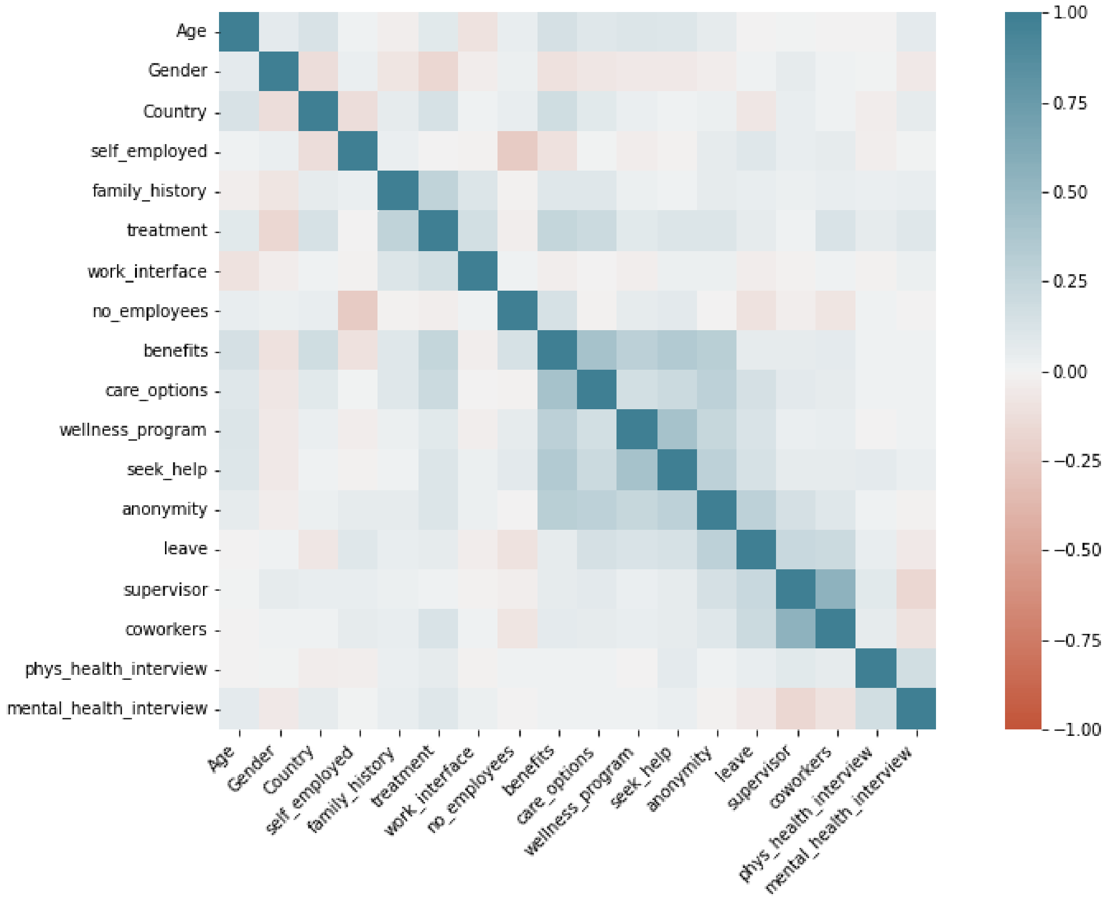
```
combined_survey2.head(5)
```

```
   Unnamed: 0       Age  ...  phys_health_interview  
mental_health_interview
0         178  0.232143  ...                      2                        
1
1         179  0.303571  ...                      2                        
1
2         180  0.392857  ...                      0                        
1
3         181  0.142857  ...                      1                        
1
4         183  0.285714  ...                      0                        
1

[5 rows x 19 columns]
```

```
np.unique(combined_survey['Country'])
```

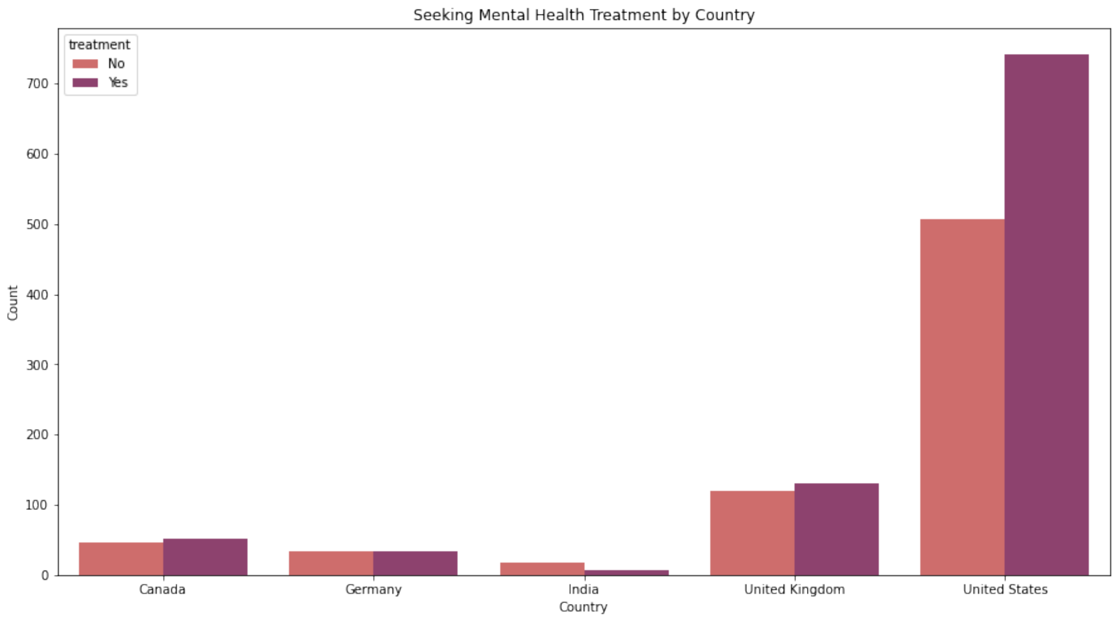
```
array(['Argentina', 'Australia', 'Austria', 'Bangladesh', 'Belarus',
       'Belgium', 'Bosnia and Herzegovina', 'Brazil', 'Bulgaria',
       'Cameroon', 'Canada', 'China', 'Colombia', 'Costa Rica',
'Croatia',
       'Czech Republic', 'Denmark', 'Egypt', 'Finland', 'France',
       'Georgia', 'Germany', 'Greece', 'Hungary', 'Iceland', 'India',
       'Indonesia', 'Ireland', 'Israel', 'Italy', 'Japan', 'Jordan',
       'Latvia', 'Malaysia', 'Mexico', 'Moldova', 'Mongolia',
       'Netherlands', 'New Zealand', 'Nigeria', 'Norway', 'Pakistan',
       'Philippines', 'Poland', 'Portugal', 'Romania', 'Russia',
'Serbia',
       'Singapore', 'Slovakia', 'Slovenia', 'South Africa', 'Spain',
       'Sri Lanka', 'Sweden', 'Switzerland', 'Taiwan', 'Thailand',
       'Turkey', 'Ukraine', 'United Kingdom', 'United States',
       'United States of America', 'Uruguay', 'Zimbabwe'],
dtype=object)
```

```
# Change Country Column United States of America to United States
combined_survey['Country'] = combined_survey['Country'].apply(lambda
x: x.replace('United States of America', 'United States'))

# Create Correlation Heatplot
import seaborn as sns
plt.figure(figsize = (15,8))
corr = combined_survey2.drop(['Unnamed: 0'], axis = 1).corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```



```
countries_count = combined_survey.groupby('Country')
['Country'].count().sort_values(ascending =
False).to_frame().rename({'Country': 'Count'}, axis = 1)
countries_count = countries_count[countries_count['Count'] >=50]

countries_list = countries_count.index
# dataset subset by countries that have a total count of 50 or more in
the data
countries_data =
combined_survey[combined_survey['Country'].isin(countries_list)]

countries_data2 = countries_data.groupby(['Country',
'treatment']).agg({'treatment':'count'}).rename({'treatment':'Count'},
axis = 1)

countries_data2 = countries_data2.reset_index()

plt.figure(figsize = (15,8))
treatment_country = sns.barplot(x = 'Country', y = 'Count', data =
countries_data2, hue = 'treatment' , palette = "flare")
treatment_country.set_title('Seeking Mental Health Treatment by
Country')
```

```
Text(0.5, 1.0, 'Seeking Mental Health Treatment by Country')
```



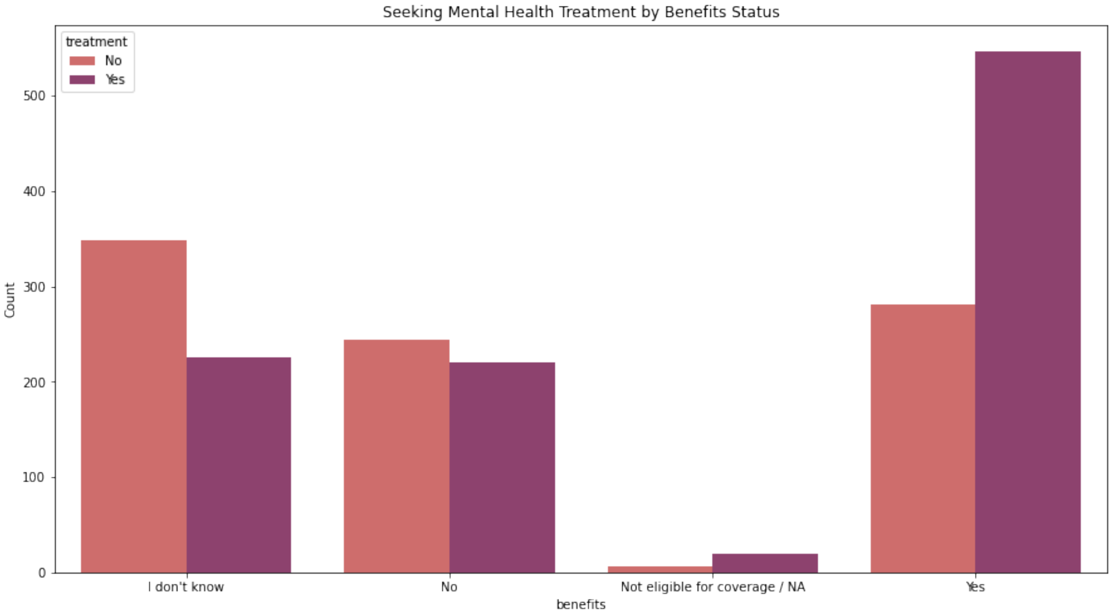
```
benefits_treatment = combined_survey.groupby(['benefits',
'treatment']).agg({'treatment':
'count'}).rename({'treatment':'Count'}, axis = 1)

benefits_treatment = benefits_treatment.reset_index()
benefits_treatment.head()
```

```
                          benefits treatment  Count
0                     I don't know        No    348
1                     I don't know       Yes    226
```

```
2                              No        No    244
3                              No       Yes    220
4  Not eligible for coverage / NA        No      6
```

```
plt.figure(figsize = (15,8))
treatment_country = sns.barplot(x = 'benefits', y = 'Count', data =
benefits_treatment, hue = 'treatment' , palette = "flare")
treatment_country.set_title('Seeking Mental Health Treatment by
Benefits Status')
```

```
Text(0.5, 1.0, 'Seeking Mental Health Treatment by Benefits Status')
```

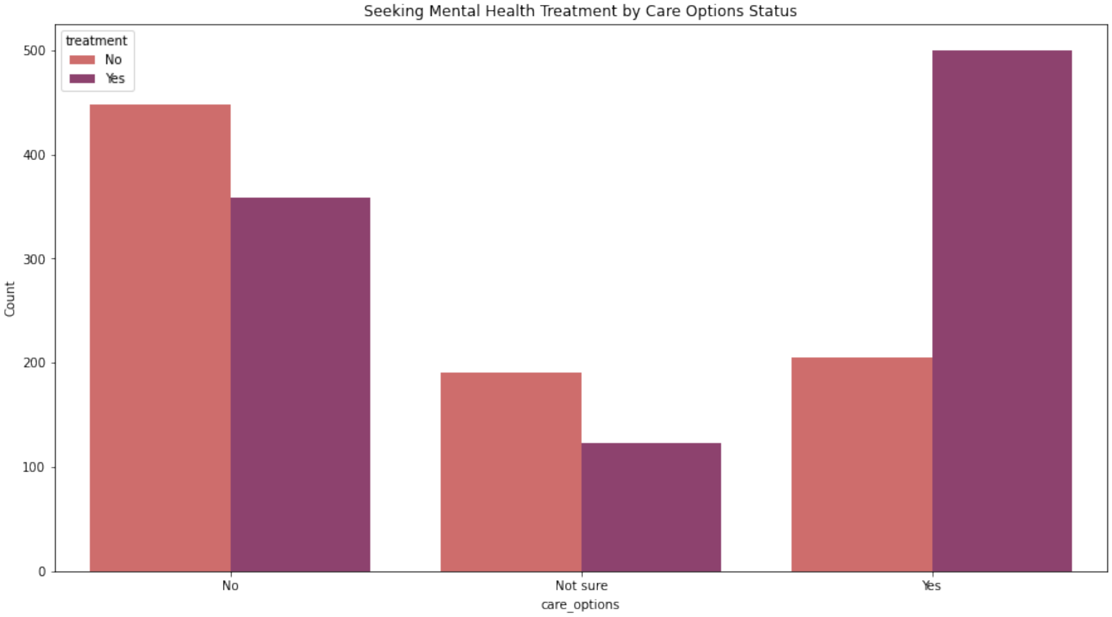


```
care_treatment = combined_survey.groupby(['care_options',
'treatment']).agg({'treatment':
'count'}).rename({'treatment':'Count'}, axis = 1)
care_treatment.head()
```

```
                        Count
care_options treatment
No           No           448
             Yes          358
Not sure     No           191
             Yes          123
Yes          No           205
```

```
care_treatment = care_treatment.reset_index()
plt.figure(figsize = (15,8))
treatment_country = sns.barplot(x = 'care_options', y = 'Count', data
= care_treatment, hue = 'treatment' , palette = "flare")
treatment_country.set_title('Seeking Mental Health Treatment by Care
Options Status')
```

```
Text(0.5, 1.0, 'Seeking Mental Health Treatment by Care Options
Status')
```


Seeking Mental Health Treatment by Care Options Status
treatment
No
Yes
Count
No
Not sure
Yes
care_options