Analyzing Risk Factors Associated with Obesity/Overweight Using Machine Learning

Phase II

DATA 606 - Capstone in Data Science
UMBC

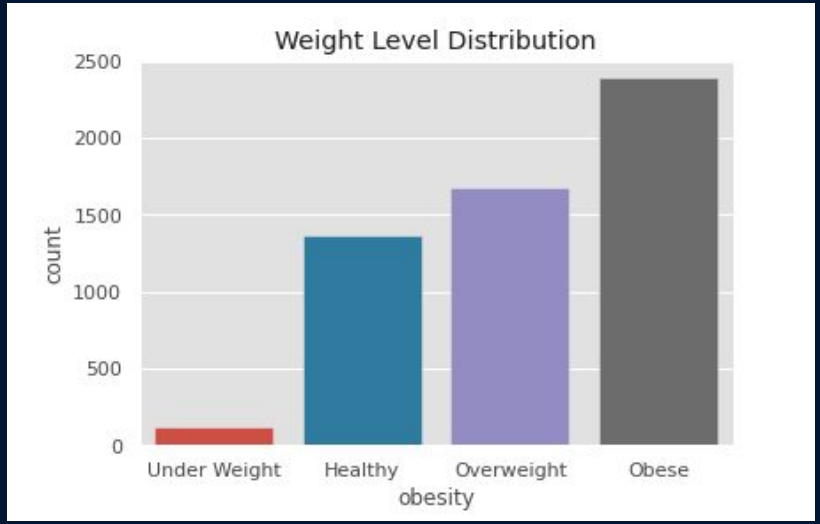
Group Members:
Sandra Pinto, Shruthi Boban, Siyu Ma

EDA & Visualizations

Added a column “obesity” showing weight level for each respondent based on CDC BMI guideline.

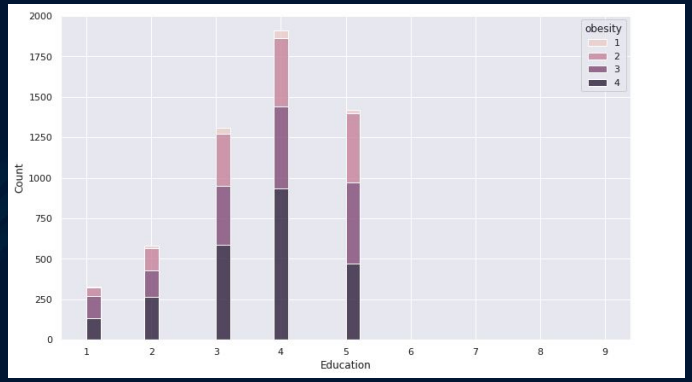
```
obese_condition = [(df_new['BMI'] < 18.5),
                   (df_new['BMI'] >= 18.5) & (df_new['BMI'] < 25.0),
                   (df_new['BMI'] >= 25.0) & (df_new['BMI'] < 30.0),
                   (df_new['BMI'] > 30.0)]

# 1 - Under Weight
# 2 - Healthy
# 3 - Overweight
# 4 - Obese
obese_value = [1, 2, 3, 4]
```

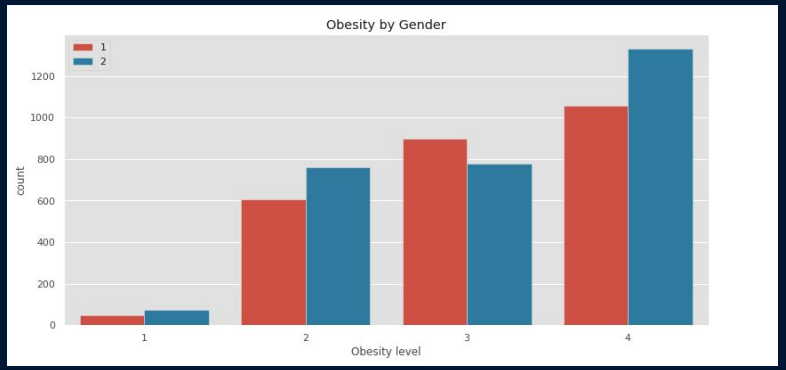
BMI	Weight Status
Below 18.5	Underweight
18.5 – 24.9	Healthy Weight
25.0 – 29.9	Overweight
30.0 and Above	Obesity

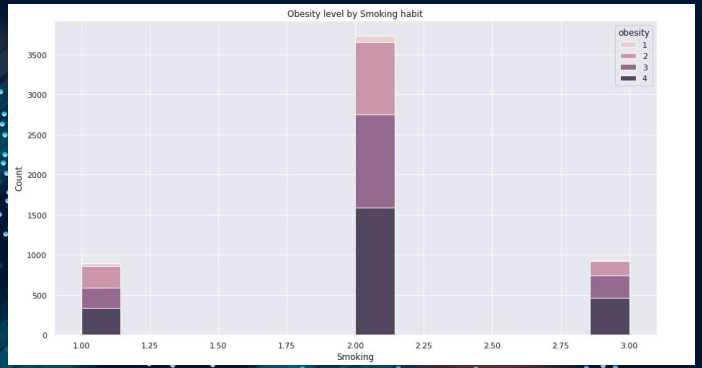


In 5556 respondents from 20~60 years old in our dataset, about 2400 respondents are obese, 1600 respondents are overweight.

EDA & Visualizations







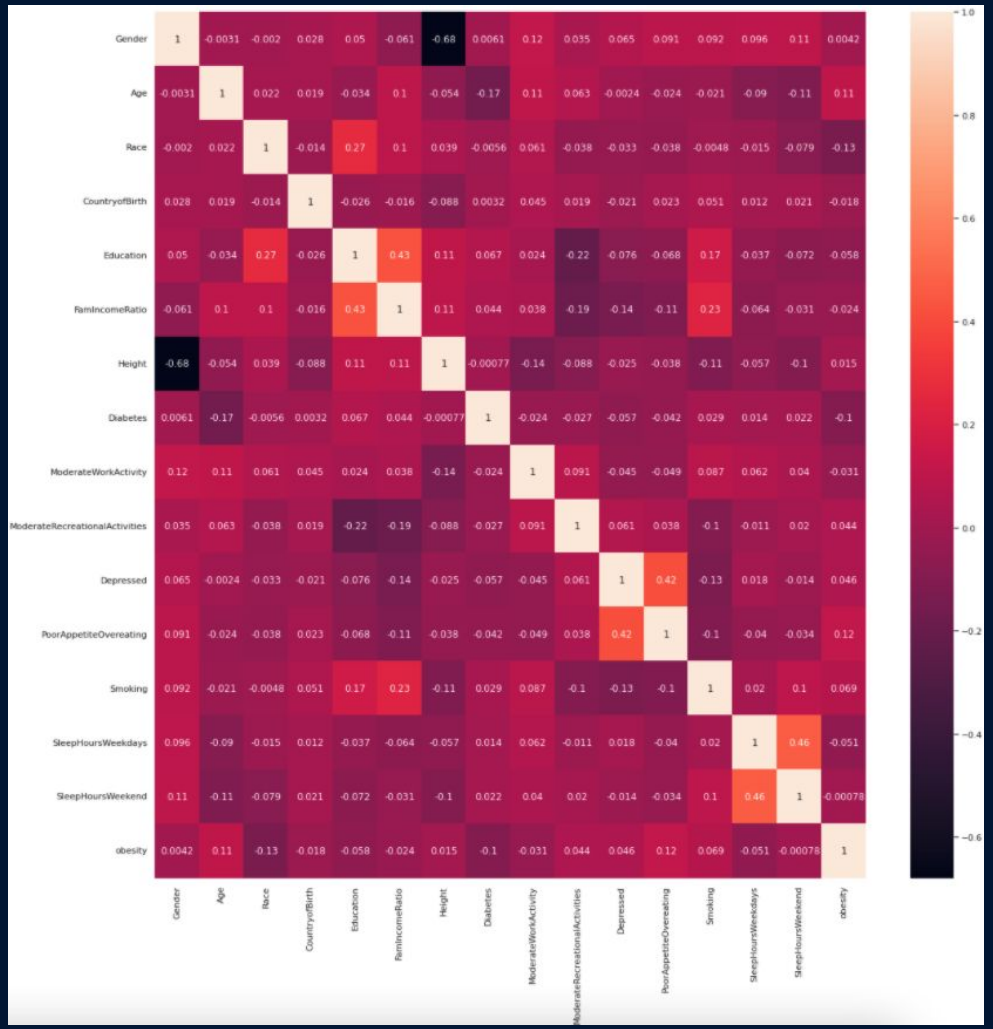
Obese group: female > male
Overweight group: female < male

1: Male
2: Female

EDA & Visualizations

Heatmap of numerical variables shows the correlation between different risk factors.

- Obesity level is highly correlated with weight and BMI
- Poor Appetite/Overeating and Age have the next two high correlation results with obesity.



Preparation & Model Construction

- Filtered out Weight and BMI from the dataset: Weight and BMI are highly correlated with obesity/overweight
- Normalizing data using mean-max transformation which scaling each variable to the range (0, 1).
- Split data to Training and Testing set.
- For our modeling section, we used Random Forest, Logistic Regression Model and SVM to predict accuracy and feature importance of risk factors

Model Construction

- We decided to create a baseline classification model as a benchmark
  - A simple model that provides reasonable results on a task or a metric you would hope any model could beat.
  - Provide the required point of comparison when evaluating all other machine learning algorithms on your problem.
  - A benchmark is vital in evaluating whether a complex model is performing well, and enables us to address the accuracy/complexity tradeoff.

```
]  # Baseline classification accuracy
   from sklearn.dummy import DummyClassifier

   baseline_classifier = DummyClassifier(strategy='most_frequent')
   baseline_classifier.fit(X_train,Y_train)
   baseline_acc = baseline_classifier.score(X_test,Y_test)
   print('Baseline accuracy = ', baseline_acc)
```

```
Baseline accuracy =  0.4738372093023256
```

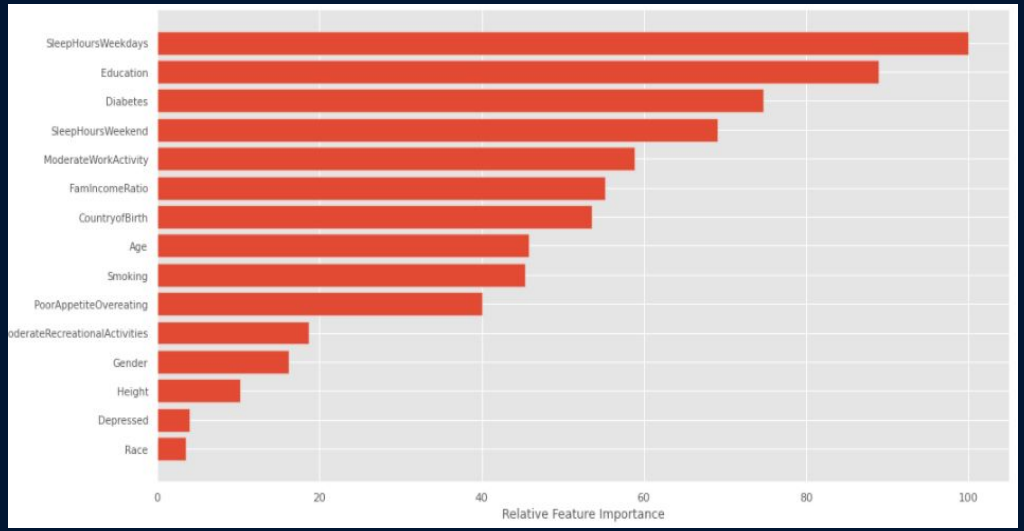
- The baseline accuracy is 47.38%, which indicates the lowest possible prediction we can get. We expecting get high accuracy from the models we selected.

Logistic Regression Model

```
#Logistic Regression Model
log_reg = LogisticRegression()
log_reg.fit(X_train,Y_train)
Y_pred = log_reg.predict(X_test)
a = accuracy_score(Y_test, Y_pred)
a
```

```
0.5203488372093024
```

```
[ ]  #Feature importance visualization
     def feature_importance_viz(model):
       feature_importance = abs(model.coef_[0])
       feature_importance = 100.0 * (feature_importance / feature_importance.max())
       sorted_idx = np.argsort(feature_importance)
       pos = np.arange(sorted_idx.shape[0]) + .5
       featfig = plt.figure(figsize=(15,7))
       featax = featfig.add_subplot(1, 1, 1)
       featax.barh(pos, feature_importance[sorted_idx], align='center')
       featax.set_yticks(pos)
       featax.set_yticklabels(np.array(X.columns)[sorted_idx], fontsize=10)
       featax.set_xlabel('Relative Feature Importance')
       plt.tight_layout()
       plt.show()
```



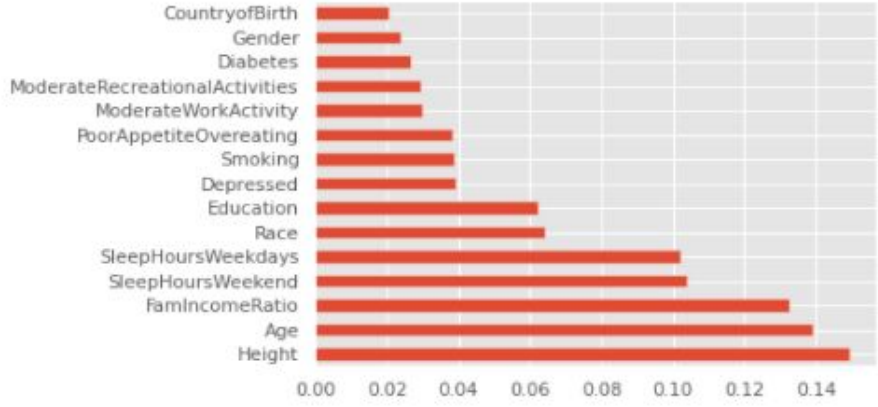
Random Forest Model

```
#Random Forest Model
rand_forest = RandomForestClassifier(n_estimators=130)
rand_forest.fit(X_train,Y_train)
Y_pred = rand_forest.predict(X_test)
a = accuracy_score(Y_test, Y_pred)
a
```

```
0.4796511627906977
```

```
feat_importances = pd.Series(rand_forest.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh', )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f897c8aca50>
```



SVM Model

```
[86] # SVM Model
     from sklearn.svm import SVC

     svm_clf = SVC(kernel='rbf')
     svm_clf.fit(X_train, Y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

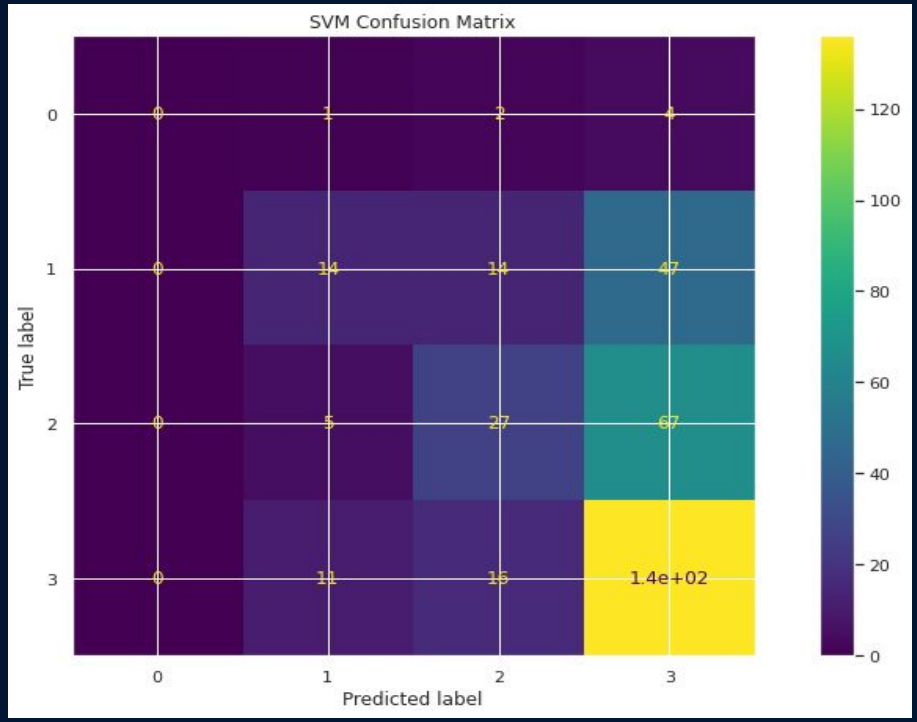
```
[87] # Predicting the Test set results
     y_pred = svm_clf.predict(X_test)
```

```
[88] from sklearn.model_selection import cross_val_score

     svm_scores = cross_val_score(svm_clf, X_train, Y_train, cv=100)
     svm_scores.mean()
```

```
0.4846153846153846
```

```
[89] accuracy_score(Y_test, y_pred)
```

```
0.5145348837209303
```



Next Steps

- Hyper parameter tuning
- Improve the R squared coefficients and confusion matrices
-

References

https://blog.ml.cmu.edu/2020/08/31/3-baselines/

https://blog.insightdatascience.com/always-start-with-a-stupid-model-no-exceptions-3a22314b9aaa