# Forecasting Diabetes

## Introduction to Problem & Data

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import HTML
```

```
# Load dataset for further analysis
df = pd.read_csv('https://raw.githubusercontent.com/Annieee-Fang/Xianchen-s-Data-Bootcamp-Stuff/refs/heads/main/NHANE
```

```
# The url of the source
source_url = 'https://archive.ics.uci.edu/dataset/887/national+health+and+nutrition+health+survey+2013-2014+(nhanes)+
```

## Preprocessing of the Dataset

```
# Extract the description of important variables from the webpage of the source
variables_table = pd.read_html(source_url)
df2 = variables_table[0]
df2
```

| | Variable Name | Role | Type | Demographic | Description | Units | Missing Values |
|---|---|---|---|---|---|---|---|
| **0** | SEQN | ID | Continuous | NaN | Respondent Sequence Number | NaN | no |
| **1** | age_group | Target | Categorical | Age | Respondent's Age Group (senior/non-senior) | NaN | no |
| **2** | RIDAGEYR | Other | Continuous | Age | Respondent's Age | NaN | no |
| **3** | RIAGENDR | Feature | Continuous | Gender | Respondent's Gender | NaN | no |
| **4** | PAQ605 | Feature | Continuous | NaN | If the respondent engages in moderate or vigor... | NaN | no |
| **5** | BMXBMI | Feature | Continuous | NaN | Respondent's Body Mass Index | NaN | no |
| **6** | LBXGLU | Feature | Continuous | NaN | Respondent's Blood Glucose after fasting | NaN | no |
| **7** | DIQ010 | Feature | Continuous | NaN | If the Respondent is diabetic | NaN | no |
| **8** | LBXGLT | Feature | Continuous | NaN | Respondent's Oral | NaN | no |
| **9** | LBXIN | Feature | Continuous | NaN | Respondent's Blood Insulin Levels | NaN | no |

```
# Create a dictionary of descriptions that are readable and short enough
new_descriptions = {
    "SEQN": "ID",
    "age_group": "Age_Group",
    "RIDAGEYR": "Age",
    "RIAGENDR": "Gender",
    "PAQ605": "Physical_Activity",
    "BMXBMI": "BMI",
    "LBXGLU": "Fasting_Glucose",
    "DIQ010": "Diabetes",
    "LBXGLT": "Oral",
    "LBXIN": "Insulin"
}

df = df.rename(columns=new_descriptions)
df
```

| | ID | Age_Group | Age | Gender | Physical_Activity | BMI | Fasting_Glucose | Diabetes | Oral | Insulin |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 73564.0 | Adult | 61.0 | 2.0 | 2.0 | 35.7 | 110.0 | 2.0 | 150.0 | 14.91 |
| **1** | 73568.0 | Adult | 26.0 | 2.0 | 2.0 | 20.3 | 89.0 | 2.0 | 80.0 | 3.85 |
| **2** | 73576.0 | Adult | 16.0 | 1.0 | 2.0 | 23.2 | 89.0 | 2.0 | 68.0 | 6.14 |
| **3** | 73577.0 | Adult | 32.0 | 1.0 | 2.0 | 28.9 | 104.0 | 2.0 | 84.0 | 16.15 |
| **4** | 73580.0 | Adult | 38.0 | 2.0 | 1.0 | 35.9 | 103.0 | 2.0 | 81.0 | 10.92 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2273** | 83711.0 | Adult | 38.0 | 2.0 | 2.0 | 33.5 | 100.0 | 2.0 | 73.0 | 6.53 |
| **2274** | 83712.0 | Adult | 61.0 | 1.0 | 2.0 | 30.0 | 93.0 | 2.0 | 208.0 | 13.02 |
| **2275** | 83713.0 | Adult | 34.0 | 1.0 | 2.0 | 23.7 | 103.0 | 2.0 | 124.0 | 21.41 |
| **2276** | 83718.0 | Adult | 60.0 | 2.0 | 2.0 | 27.4 | 90.0 | 2.0 | 108.0 | 4.99 |
| **2277** | 83727.0 | Adult | 26.0 | 1.0 | 2.0 | 24.5 | 108.0 | 2.0 | 108.0 | 3.76 |

2278 rows × 10 columns

```
df2['Description'].str.strip()
# Display the Original Variable Name,   Descriptions that we are using, and the original Detailed Description
df2=df2.rename({'Variable Name': 'Original Variable Name', 'Description': 'Detailed Description'}, axis = 1)
df2['Description'] = df2['Original Variable Name'].map(new_descriptions)
pd.concat((df2['Description'], df2['Detailed Description']), axis = 1)
```

| | Description | Detailed Description |
|---|---|---|
| **0** | ID | Respondent Sequence Number |
| **1** | Age_Group | Respondent's Age Group (senior/non-senior) |
| **2** | Age | Respondent's Age |
| **3** | Gender | Respondent's Gender |
| **4** | Physical_Activity | If the respondent engages in moderate or vigor... |
| **5** | BMI | Respondent's Body Mass Index |
| **6** | Fasting_Glucose | Respondent's Blood Glucose after fasting |
| **7** | Diabetes | If the Respondent is diabetic |
| **8** | Oral | Respondent's Oral |
| **9** | Insulin | Respondent's Blood Insulin Levels |

```
#exchange 2's with 3's
df['Diabetes'] = df['Diabetes'].replace({2.0: 3.0, 3.0: 2.0})
df['Diabetes'].value_counts()
```

| | count |
|---|---|
| **Diabetes** | |
| **3.0** | 2199 |
| **2.0** | 58 |
| **1.0** | 21 |

**dtype:** int64

```
df = df.drop('ID', axis=1)
```

```
df
```

| | Age_Group | Age | Gender | Physical_Activity | BMI | Fasting_Glucose | Diabetes | Oral | Insulin |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Adult | 61.0 | 2.0 | 2.0 | 35.7 | 110.0 | 3.0 | 150.0 | 14.91 |
| **1** | Adult | 26.0 | 2.0 | 2.0 | 20.3 | 89.0 | 3.0 | 80.0 | 3.85 |
| **2** | Adult | 16.0 | 1.0 | 2.0 | 23.2 | 89.0 | 3.0 | 68.0 | 6.14 |
| **3** | Adult | 32.0 | 1.0 | 2.0 | 28.9 | 104.0 | 3.0 | 84.0 | 16.15 |
| **4** | Adult | 38.0 | 2.0 | 1.0 | 35.9 | 103.0 | 3.0 | 81.0 | 10.92 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2273** | Adult | 38.0 | 2.0 | 2.0 | 33.5 | 100.0 | 3.0 | 73.0 | 6.53 |
| **2274** | Adult | 61.0 | 1.0 | 2.0 | 30.0 | 93.0 | 3.0 | 208.0 | 13.02 |
| **2275** | Adult | 34.0 | 1.0 | 2.0 | 23.7 | 103.0 | 3.0 | 124.0 | 21.41 |
| **2276** | Adult | 60.0 | 2.0 | 2.0 | 27.4 | 90.0 | 3.0 | 108.0 | 4.99 |
| **2277** | Adult | 26.0 | 1.0 | 2.0 | 24.5 | 108.0 | 3.0 | 108.0 | 3.76 |

2278 rows × 9 columns

## Preliminary Examination of the Dataset

### Descriptive Statistics

```
df['Diabetes'].value_counts()
```

| Diabetes | count |
|---|---|
| **3.0** | 2199 |
| **2.0** | 58 |
| **1.0** | 21 |

**dtype:** int64

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2278 entries, 0 to 2277
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Age_Group          2278 non-null   object 
 1   Age                2278 non-null   float64
 2   Gender             2278 non-null   float64
 3   Physical_Activity  2278 non-null   float64
 4   BMI                2278 non-null   float64
 5   Fasting_Glucose    2278 non-null   float64
 6   Diabetes           2278 non-null   float64
 7   Oral               2278 non-null   float64
 8   Insulin            2278 non-null   float64
dtypes: float64(8), object(1)
memory usage: 160.3+ KB
```

```
df #.head(10)
```

| | Age_Group | Age | Gender | Physical_Activity | BMI | Fasting_Glucose | Diabetes | Oral | Insulin |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Adult | 61.0 | 2.0 | 2.0 | 35.7 | 110.0 | 3.0 | 150.0 | 14.91 |
| **1** | Adult | 26.0 | 2.0 | 2.0 | 20.3 | 89.0 | 3.0 | 80.0 | 3.85 |
| **2** | Adult | 16.0 | 1.0 | 2.0 | 23.2 | 89.0 | 3.0 | 68.0 | 6.14 |
| **3** | Adult | 32.0 | 1.0 | 2.0 | 28.9 | 104.0 | 3.0 | 84.0 | 16.15 |
| **4** | Adult | 38.0 | 2.0 | 1.0 | 35.9 | 103.0 | 3.0 | 81.0 | 10.92 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2273** | Adult | 38.0 | 2.0 | 2.0 | 33.5 | 100.0 | 3.0 | 73.0 | 6.53 |
| **2274** | Adult | 61.0 | 1.0 | 2.0 | 30.0 | 93.0 | 3.0 | 208.0 | 13.02 |
| **2275** | Adult | 34.0 | 1.0 | 2.0 | 23.7 | 103.0 | 3.0 | 124.0 | 21.41 |
| **2276** | Adult | 60.0 | 2.0 | 2.0 | 27.4 | 90.0 | 3.0 | 108.0 | 4.99 |
| **2277** | Adult | 26.0 | 1.0 | 2.0 | 24.5 | 108.0 | 3.0 | 108.0 | 3.76 |

2278 rows × 9 columns

```
df['Age_Group'].value_counts()
```

| Age_Group | count |
|---|---|
| **Adult** | 1914 |
| **Senior** | 364 |

**dtype:** int64

```
# The number of people of each ages tested
df['Age'].value_counts().sort_index(ascending=True)
```

| Age | count |
|---|---|
| **12.0** | 58 |
| **13.0** | 49 |
| **14.0** | 60 |
| **15.0** | 48 |
| **16.0** | 70 |
| **...** | ... |
| **76.0** | 13 |
| **77.0** | 9 |
| **78.0** | 8 |
| **79.0** | 8 |
| **80.0** | 107 |

69 rows × 1 columns

**dtype:** int64

```
age_sorted_data = df.sort_values(by='Age', ascending=True)[df['Age_Group'] == 'Adult']
age_sorted_data
```

```
<ipython-input-15-c0978ea22d72>:1: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  age_sorted_data = df.sort_values(by='Age', ascending=True)[df['Age_Group'] == 'Adult']
```

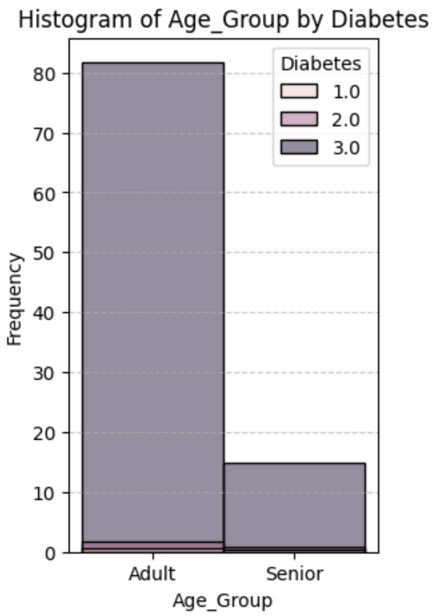
| | Age_Group | Age | Gender | Physical_Activity | BMI | Fasting_Glucose | Diabetes | Oral | Insulin |
|---|---|---|---|---|---|---|---|---|---|
| **253** | Adult | 12.0 | 1.0 | 2.0 | 20.5 | 100.0 | 3.0 | 100.0 | 13.18 |
| **268** | Adult | 12.0 | 2.0 | 2.0 | 20.0 | 115.0 | 3.0 | 140.0 | 14.41 |
| **270** | Adult | 12.0 | 2.0 | 2.0 | 27.5 | 94.0 | 3.0 | 100.0 | 14.67 |
| **802** | Adult | 12.0 | 2.0 | 2.0 | 17.1 | 94.0 | 3.0 | 101.0 | 8.30 |
| **808** | Adult | 12.0 | 1.0 | 2.0 | 16.1 | 96.0 | 3.0 | 197.0 | 8.57 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **82** | Adult | 64.0 | 2.0 | 2.0 | 25.4 | 90.0 | 3.0 | 215.0 | 2.10 |
| **2214** | Adult | 64.0 | 2.0 | 2.0 | 23.7 | 110.0 | 2.0 | 82.0 | 9.65 |
| **403** | Adult | 64.0 | 1.0 | 2.0 | 27.5 | 99.0 | 3.0 | 119.0 | 4.47 |
| **948** | Adult | 64.0 | 2.0 | 2.0 | 41.5 | 91.0 | 3.0 | 149.0 | 15.52 |
| **1299** | Adult | 64.0 | 1.0 | 2.0 | 25.0 | 98.0 | 3.0 | 103.0 | 5.71 |

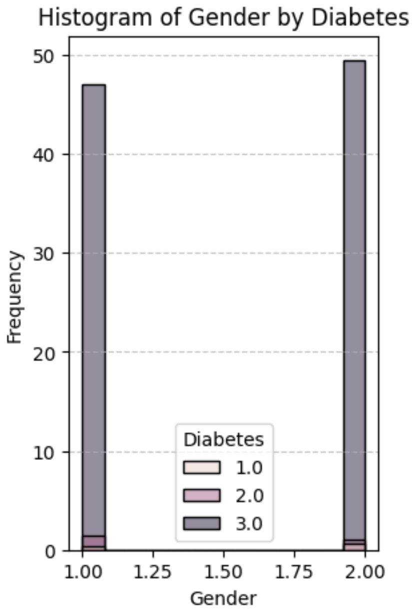
1914 rows × 9 columns

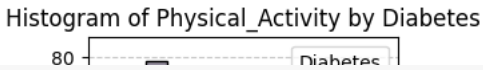
## Initial Visualizations

```
variables1 = ['Age_Group', 'Gender', 'Physical_Activity', 'Oral']
```

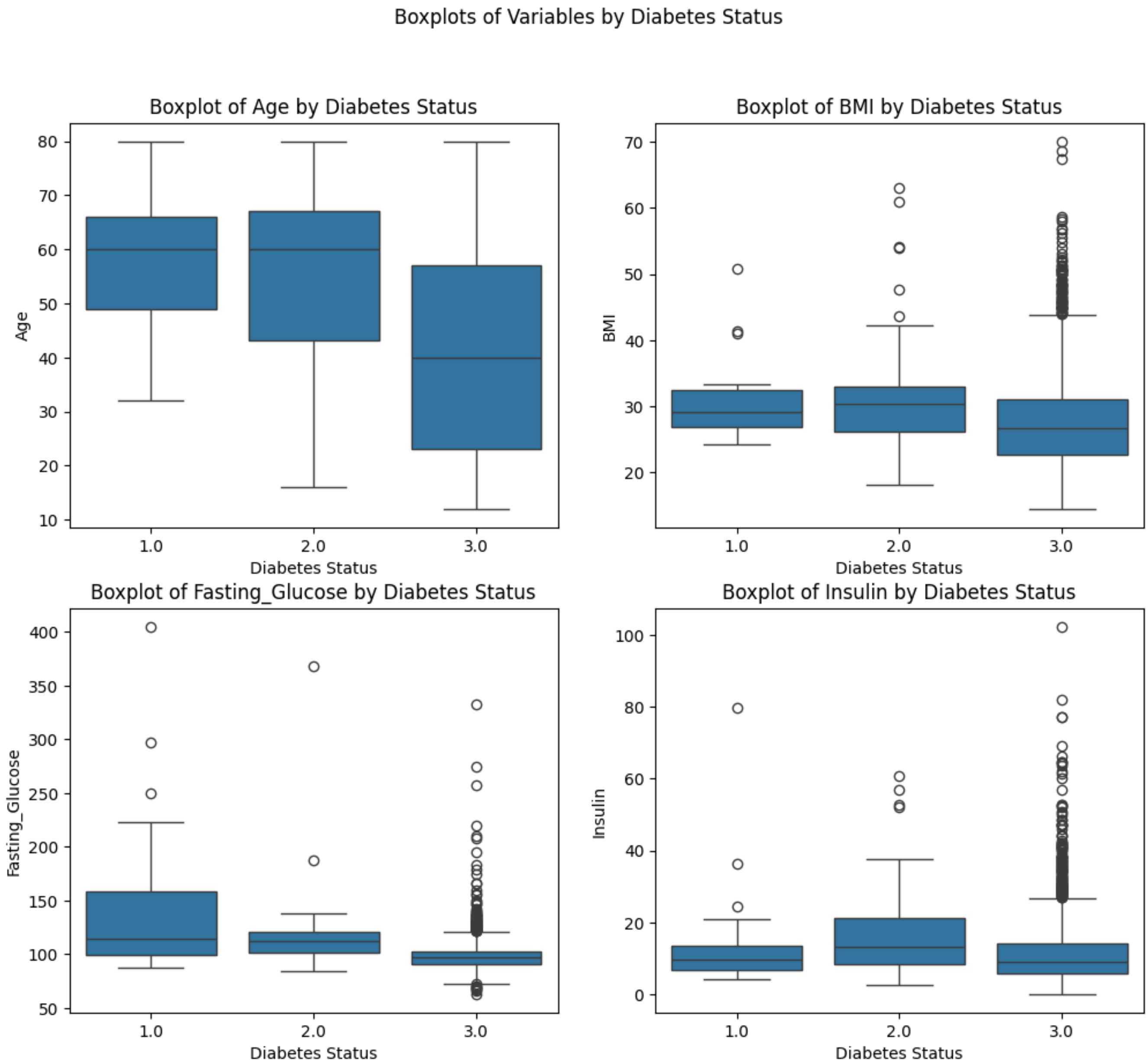
```
for col in variables1:
    plt.figure(figsize=(3, 5))
    sns.histplot(data=df, x=col, hue='Diabetes', kde=False, stat="percent")
    t = f'Histogram of {col} by Diabetes'
    plt.title(t)
    plt.savefig(t+'.png')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
```




```
variables2 = ['Age', 'BMI', 'Fasting_Glucose', 'Insulin']
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
index=0
for i in range(2):
    for j in range(2):
        sns.boxplot(x='Diabetes', y=variables2[index], data=df, ax=axes[i][j])
        axes[i][j].set_title(f'Boxplot of {variables2[index]} by Diabetes Status')
        axes[i][j].set_xlabel('Diabetes Status')
        axes[i][j].set_ylabel(variables2[index])
        index+=1
plt.suptitle('Boxplots of Variables by Diabetes Status');
plt.savefig('Boxplots of Variables by Diabetes Status.png')
```




## Modeling & Interpretations

```
X = df.drop(columns=['Diabetes'])
y = df['Diabetes']
```

### Baseline Model

```
from sklearn.metrics import mean_squared_error
```

```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
```

```
y.value_counts(normalize = True)
```

| | proportion |
|---|---|
| **Diabetes** | |
| **3.0** | 0.965320 |
| **2.0** | 0.025461 |
| **1.0** | 0.009219 |

**dtype:** float64

## Logistics Models

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
```

```
ohe = OneHotEncoder(drop='first')
sscaler = StandardScaler()
transformer = make_column_transformer((ohe, ['Age_Group']), remainder=sscaler)

lr = LogisticRegression(solver='lbfgs', max_iter=200)
pipe = Pipeline([('transform', transformer), ('logreg', lr)])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

param_grid = [{'logreg__C': 10**np.linspace(-3, 3, 10)}]
```

```
from sklearn.metrics import make_scorer
```

```
from sklearn.metrics import precision_score #, accuracy_score, recall_score
```

```
methods_score = {}
new_precision = make_scorer(precision_score, average='macro', zero_division=1) # there existed some errors while us:
metrics= ["accuracy", new_precision]
for metric in metrics:
  lr_gridsearch_accuracy = GridSearchCV(pipe, param_grid, cv=10, scoring=metric)
  lr_gridsearch_accuracy.fit(X_train, y_train)
  score_accuracy = lr_gridsearch_accuracy.best_score_
  methods_score[metric] = score_accuracy
methods_score['precision'] = methods_score[new_precision]
```

```
methods_score.pop(new_precision)
print(methods_score)
```

```
{'accuracy': 0.9648771993034287, 'precision': 0.9881092495846595}
```

## Decision Tree Regression Model

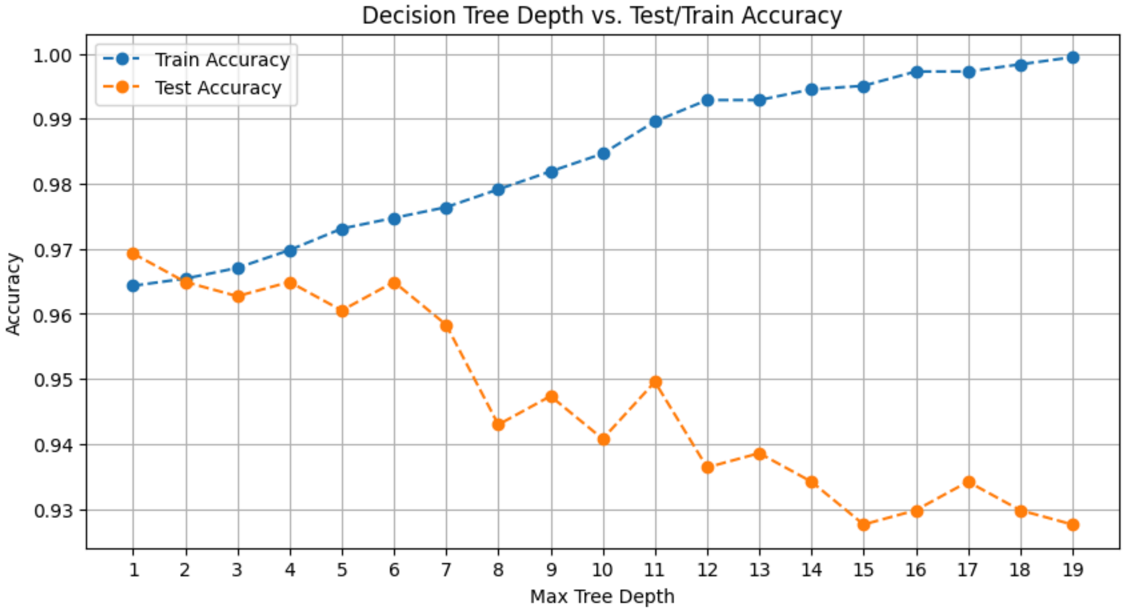
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
from sklearn.inspection import DecisionBoundaryDisplay
```

```
ohe = OneHotEncoder(drop='first')
sscaler = StandardScaler()
transformer = make_column_transformer((ohe, ['Age_Group']), remainder=sscaler)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train_encoded = transformer.fit_transform(X_train)
X_test_encoded = transformer.transform(X_test)
```

```
train_scores = []
test_scores = []

for d in range(1, 20):
    dtree = DecisionTreeClassifier(max_depth=d, random_state=42)
    dtree.fit(X_train_encoded, y_train)

    y_train_preds = dtree.predict(X_train_encoded)
    y_test_preds = dtree.predict(X_test_encoded)

    train_scores.append(accuracy_score(y_train, y_train_preds))
    test_scores.append(accuracy_score(y_test, y_test_preds))
```

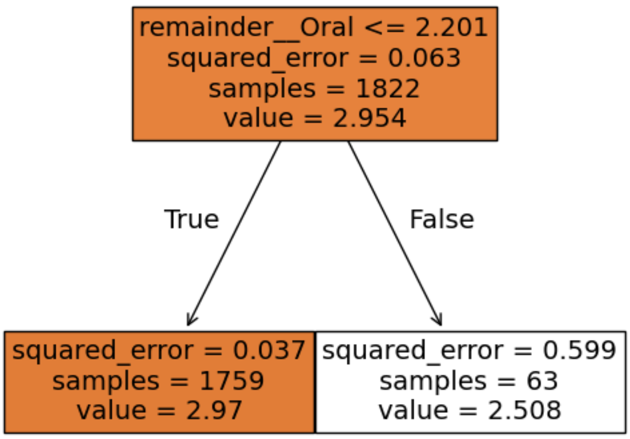
```
plt.figure(figsize=(10, 5))
plt.plot(range(1, 20), train_scores, '--o', label='Train Accuracy')
plt.plot(range(1, 20), test_scores, '--o', label='Test Accuracy')
plt.grid()
plt.legend()
plt.xticks(range(1, 20))
```

```
plt.xlabel('Max Tree Depth')
plt.ylabel('Accuracy')
plt.title('Decision Tree Depth vs. Test/Train Accuracy')
plt.savefig('Decision Tree Depth vs. Test and Train Accuracy.png')
```




```
from sklearn.tree import DecisionTreeRegressor
#fit a decision tree model with a max depth = 1 (lowest mse test score on graph)
dtree = DecisionTreeRegressor(max_depth = 1).fit(X_train_encoded, y_train)
```

```
plt.figure(figsize=(6,6))
plot_tree(dtree, filled=True, feature_names=transformer.get_feature_names_out().tolist(), fontsize=14)
plt.savefig('Decision Process.png');
```




```
max(test_scores)
```

```
0.9692982456140351
```

## Random Forest Regression Model

```
from sklearn.ensemble import RandomForestClassifier
```

```
train_scores = []
test_scores = []

for n in range(10, 110, 10): # n is the numbers of estimators
```

```
    rf = RandomForestClassifier(n_estimators=n, random_state=42
    rf.fit(X_train_encoded, y_train)
    y_train_preds = rf.predict(X_train_encoded)
    y_test_preds = rf.predict(X_test_encoded)
    train_scores.append(accuracy_score(y_train, y_train_preds))
    test_scores.append(accuracy_score(y_test, y_test_preds))
```