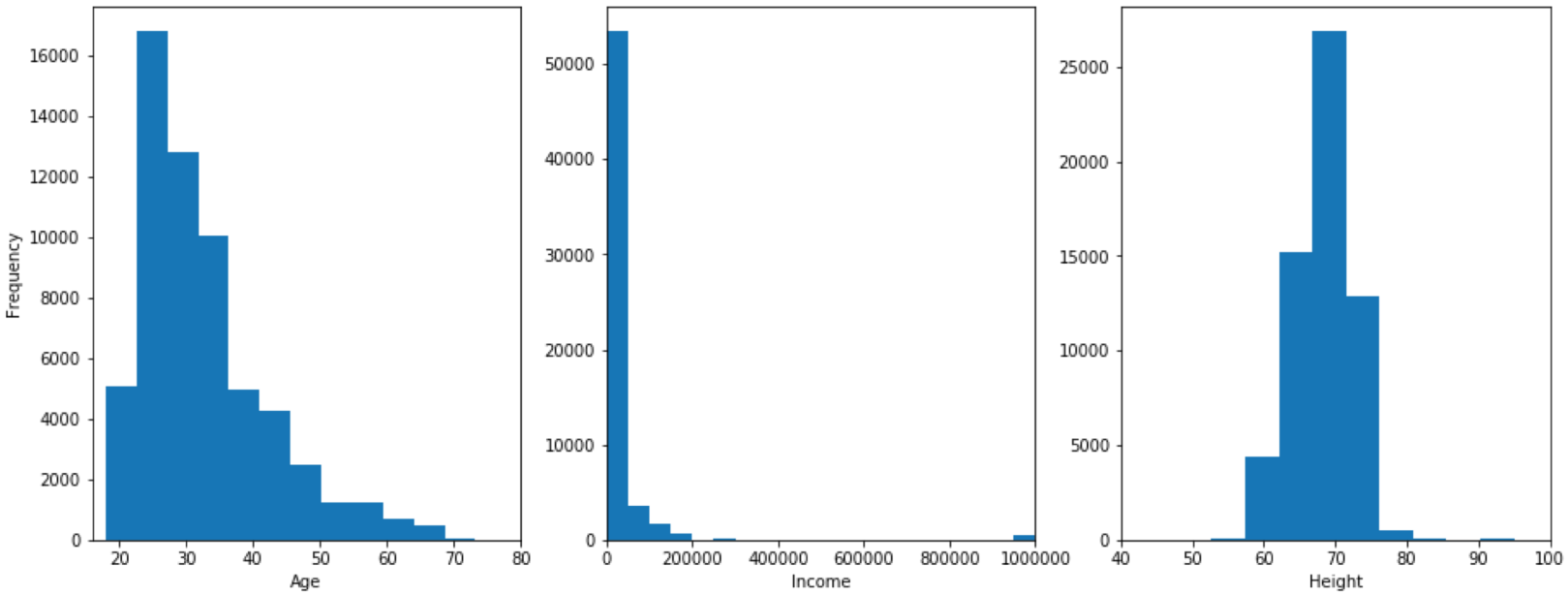
Date-A-Scientist

Capstone Project – Machine Learning Fundamentals

Prepared by Magnus Wangensteen – Nov 19th 2018

Inspection of Data

Inspection of numerical features in the dataset.



Data preconditioning

- Replace NaNs with appropriate values.
  - NaN heights replaced with mean height (*df.height = df.height.fillna(round(df.height.mean(),1))*)
  - Other NaN features filled with «rather not say» where appropriate
- Some features mapped to segments: drinks, religion, age, body_type, diet, education, job, kids, pets, sign
- Mapping of all features (or new features segments) to numerical categories (normally "rather not say" to zero)

Features Mapping Example

Extracting first word from sign feature using a lambda function. Returns zodiac sign only.

Mapping to numerical value.

”rather not say” is mapped to zero.

```
sign_mapping = lambda sign: sign.split(' ')[0] if not sign=="rather not say" else "rather not say"
```

```
df["sign_segment"] = df["sign"].astype(str).apply(sign_mapping)
```

```
def valuemapping(values_to_map):
    values = values_to_map.unique()
    mapping = {}
    shift = 0
    for i, value in enumerate(values, 1):
        mapping[value] = i - shift
        if value == "rather not say":
            mapping[value] = 0
            shift = 1
    return mapping
```

```
sign_mapping = valuemapping(df.sign_segment)
print(sign_mapping)
df["sign_code"] = df.sign_segment.map(sign_mapping)
```

```
{'gemini': 1, 'cancer': 2, 'pisces': 3, 'aquarius': 4, 'taurus': 5, 'virgo': 6, 'sagittarius': 7, 'leo': 8, 'rather n
ot say': 0, 'aries': 9, 'libra': 10, 'scorpio': 11, 'capricorn': 12}
```

```
df.sign_code.value_counts()
```

```
0     11056
8      4374
1      4310
10     4207
2      4206
6      4141
5      4140
11     4134
9      3989
3      3946
7      3942
4      3928
12     3573
Name: sign_code, dtype: int64
```

Essay Analysis Function

New features created:

'text_length'

'avg_word_length'

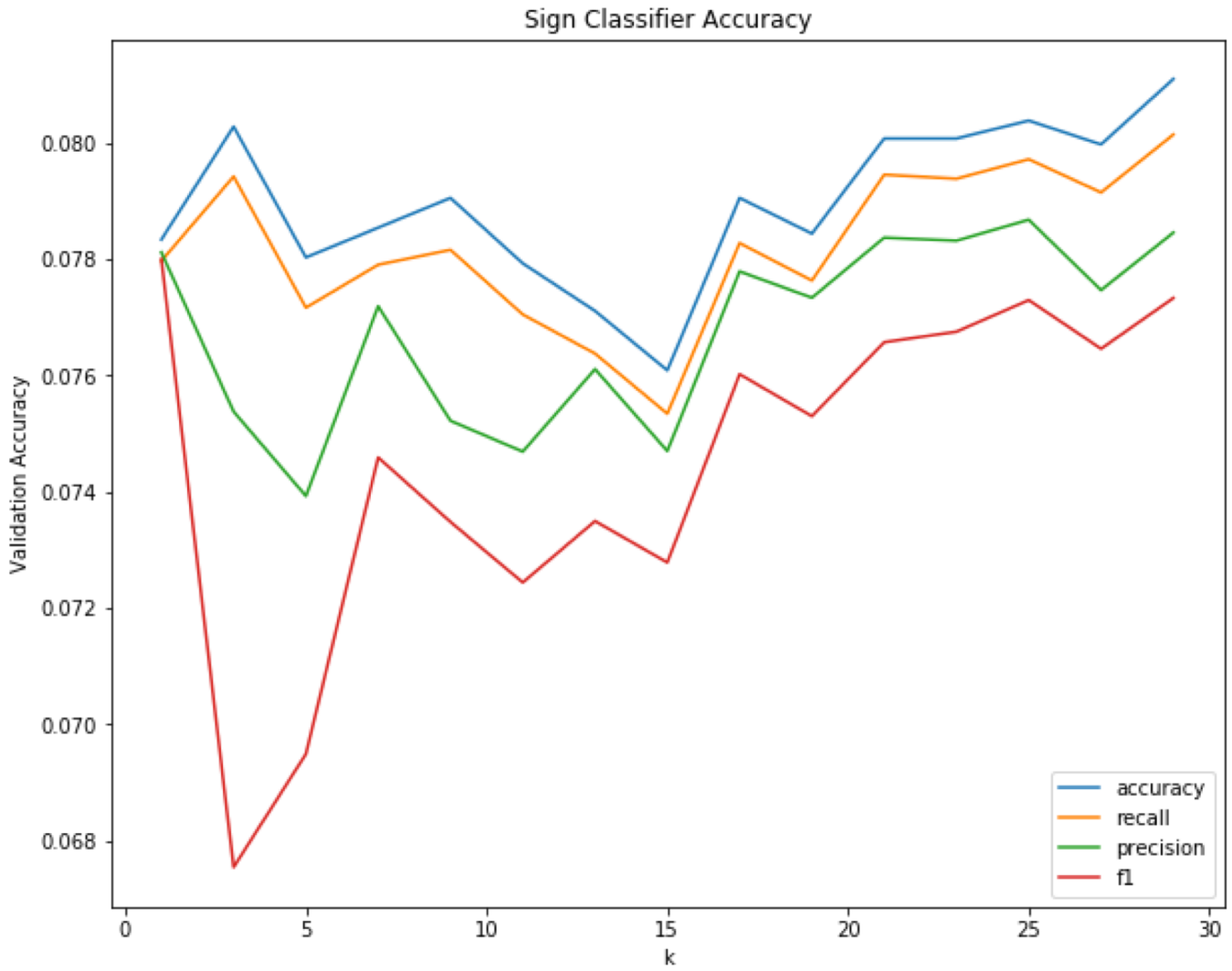
'i_me_frequency'

```python
def essayanalysis(text):
    text = text.replace("<br />", ' ')
    text = text.replace("\n", ' ')
    for char in '-.,:?/+!':
        text=text.replace(char,' ')
    text = text.lower()
    word_list = text.split()
    #print(word_list)
    text_length = len(word_list)
    wordlength = []
    imecount = 0
    iwords = ["i", "i'm", "me"]
    for word in word_list:
        wordlength.append(len(word))
        if word in iwords:
            imecount +=1
            #print(word)
    if len(word_list) != 0:
        avgwordlength = sum(wordlength) / len(word_list)
    else:
        avgwordlength = 0
    #return text_length#, avgwordlength, imecount
    return pd.Series([text_length, avgwordlength, imecount])
```

```python
essay_selection = 'all'
if essay_selection == 'all':
    df[['text_length','avg_word_length', 'i_me_count']] = df.apply(lambda row: essayanalysis(row['essay0'] + ' '
                                                                                             + row['essay1'] + ' '
                                                                                             + row['essay2'] + ' '
                                                                                             + row['essay3'] + ' '
                                                                                             + row['essay4'] + ' '
                                                                                             + row['essay5'] + ' '
                                                                                             + row['essay6'] + ' '
                                                                                             + row['essay7'] + ' '
                                                                                             + row['essay8'] + ' '
                                                                                             + row['essay9']), axis
elif essay_selection == 'first4':
    df[['text_length','avg_word_length', 'i_me_count']] = df.apply(lambda row: essayanalysis(row['essay0'] + ' '
                                                                                             + row['essay1'] + ' '
                                                                                             + row['essay2'] + ' '
                                                                                             + row['essay3']), axis
elif essay_selection == 'first1':
    df[['text_length','avg_word_length', 'i_me_count']] = df.apply(lambda row: essayanalysis(row['essay0']), axis=1

```

```python
df['i_me_frequency'] = df.apply((lambda row: row['i_me_count']/row['text_length'] if row['text_length'] != 0
                                 else 0), axis=1)
```



KNN Classifier – Zodiac sign prediction

Plot shows accuracy, recall, precission and f1 for different number of neighbours (k) along the x-axis.

Most of the features in the dataset were used as input to the knn classifier.

We observe that there is little convergence and that highest accuracy is a bit above 8%. Which is what we would get by random guessing of a zodiac sign (1/12 chance of guessing correct).

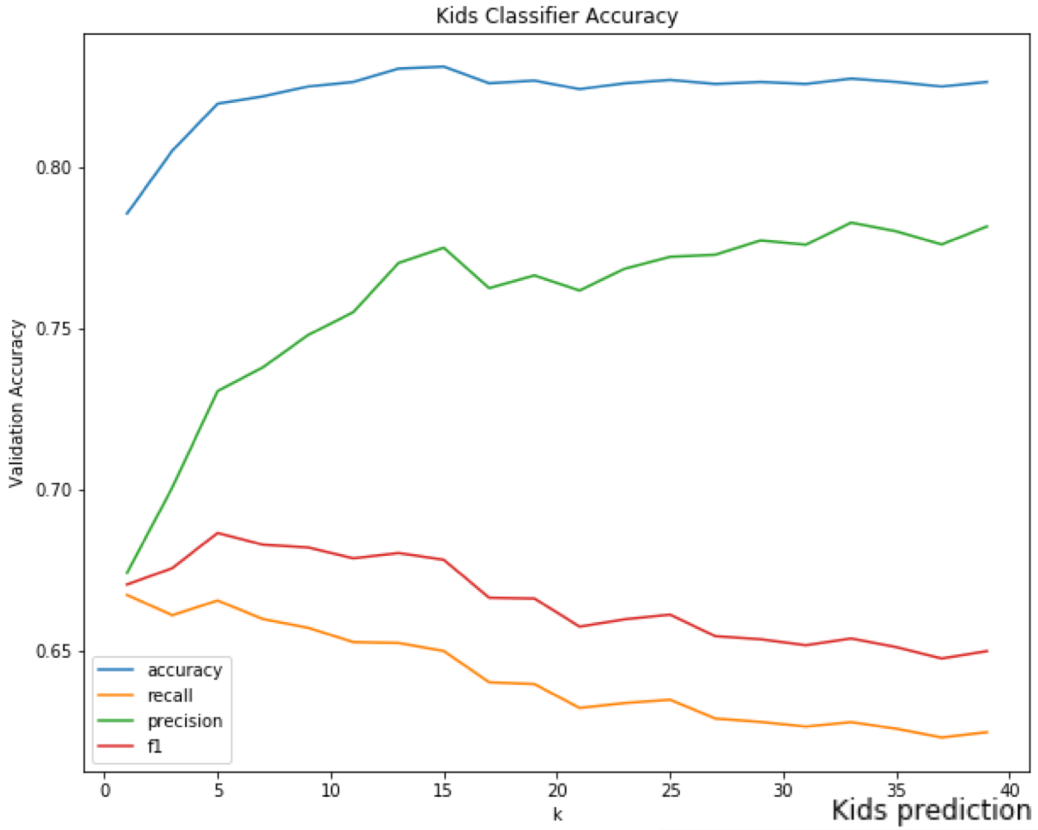
Bad news for those who believe in astrology.

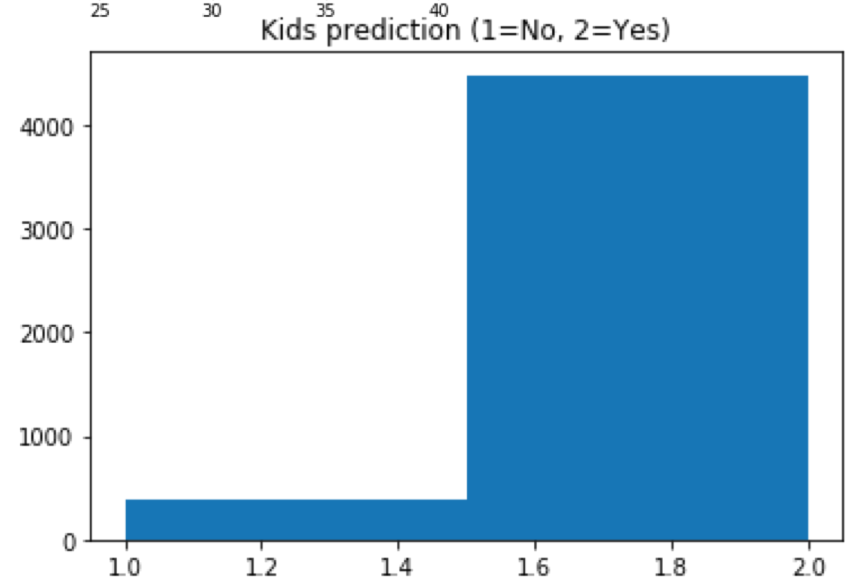
KNN Classifier – Kids prediction

Kids prediction (segmented into 'yes' or 'no') is done after zeros ('rather not tell') are removed from the data.

The classifier estimates a lot more 2 (yes) than 1 (no). This is plotted to verify that it doesn't only predict the most occurring answer in the training labels, which would be 2.

Accuracy is high, more than 80%. Also recall is decent.





KNN Classifier – Smokes prediction

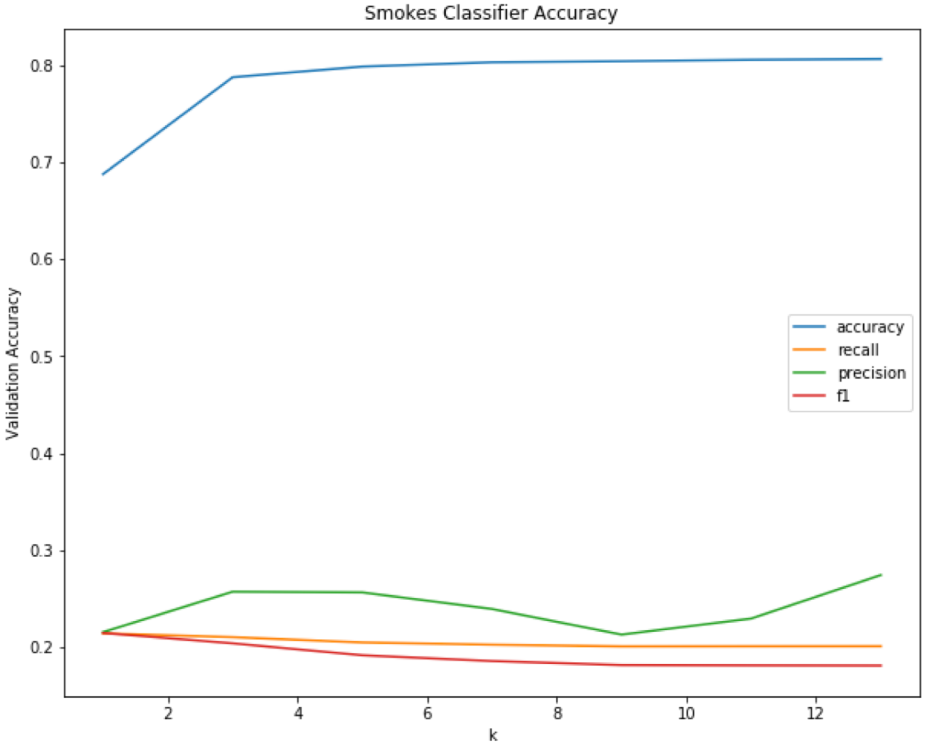
Smokes prediction using the original feature categories.

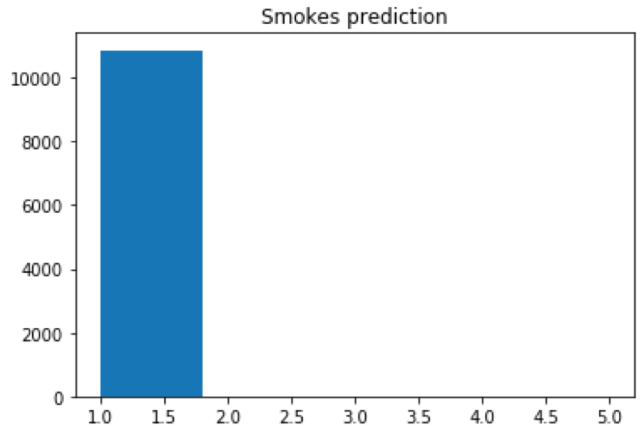
We observe that the precision is high, at 80%, but inspecting the predicted value, we see that the classifier always predicts a non-smoker. This seems rational since there are a lot more counts in the non-smoker segment (1) than in any other segment ('yes', 'sometimes', 'when drinking', 'trying to quit'). Hence, accuracy is high by always predicting a non-smoker, but recall, precision and f1 are low.

```
1  smokes_prediction_data.smokes_code.value_counts()
```

```
1    43896
3     3787
2     3040
5     2231
4     1480
Name: smokes_code, dtype: int64
```

In the next test, the smokes segments are merged into one before the classification.





KNN Classifier – Smokes Prediction II

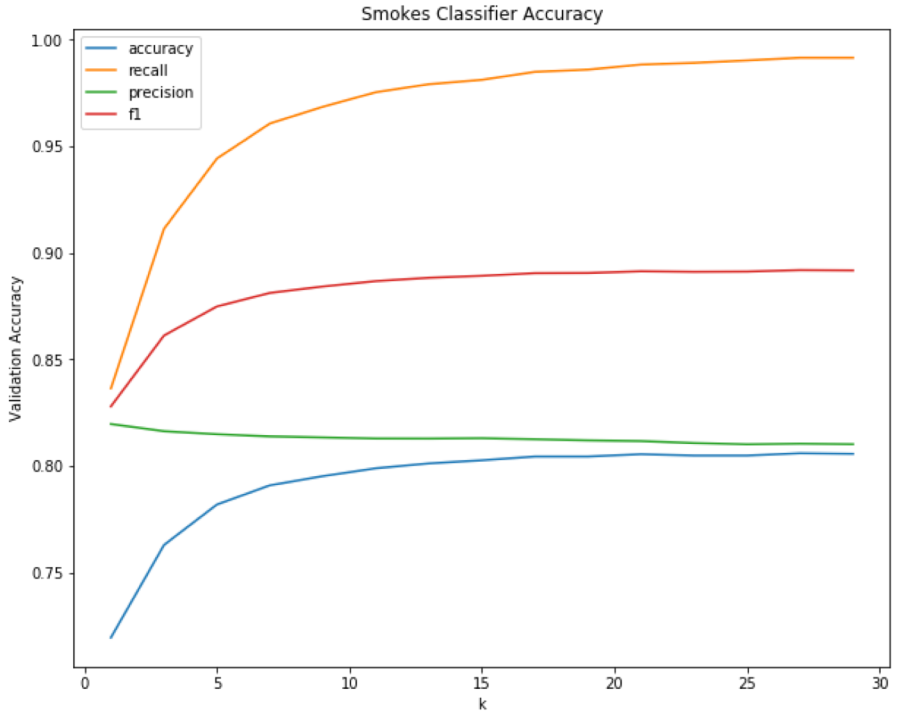
In the experiment where all smokers are classified into one bin (see code below), the accuracy is similar to the previous experiment, but recall, precision and f1 score is much higher. I.e. reduced number of false negatives.

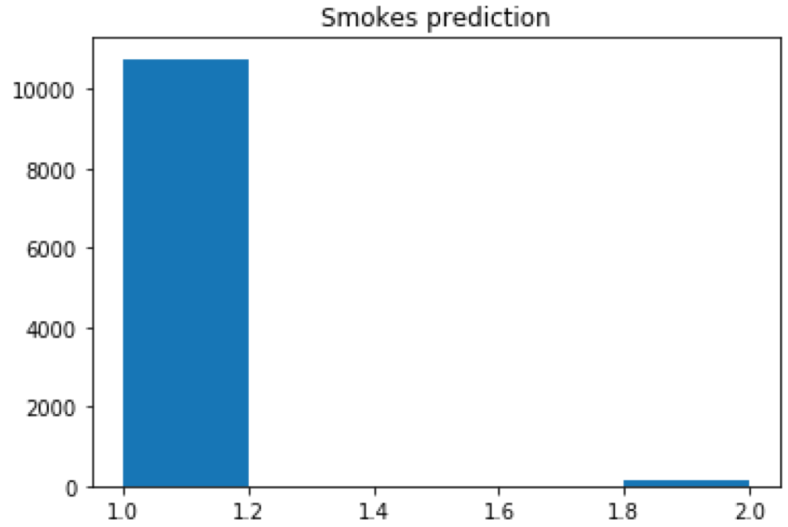
```
1  smokes_prediction_labels_bin = smokes_prediction_labels.apply(lambda x: 1 if x==1 else 2)
```

```
1  smokes_prediction_labels_bin.value_counts()
```

```
1    43896
2    10538
Name: smokes_code, dtype: int64
```

Also, we see that predictions are made for both 1 (non-smoker and 2 (smoker)





SVC Classifier

Both SVC linear and SVC rbf were used to attempt to classify if the person has kids or not.

We observe that both the linear and the rbf kernel does the prediction with same accuracy, recall and precision. Accuracy is similar as the KNN classifier, but the SVC classifier training is much slower for many input features. Hence, the features used are the 4 with the highest absolute .corr() value.

We also observe that both 1 (no kids) and 2 (kids) are predicted.

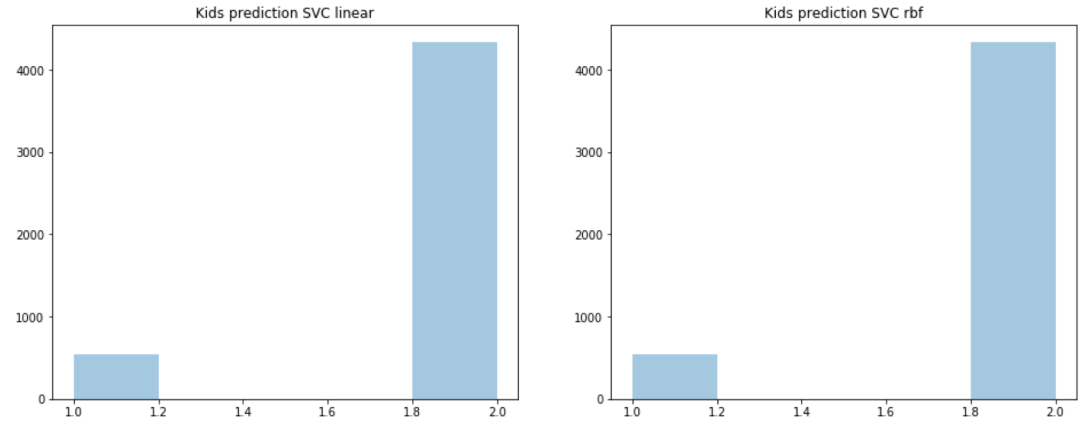
```
print("Accuracy: ", kids_accuracies_SVClin)
print("Recall: ", kids_recalls_SVClin)
print("Precision: ", kids_precisions_SVClin)
print("F1 score: ", kids_f1_SVClin)
```

```
Accuracy:  [0.837605085093295]
Recall:  [0.6691488741990224]
Precision:  [0.7837857074262268]
F1 score:  [0.6990737047323206]
```

```
print("Accuracy: ", kids_accuracies_SVCrbf)
print("Recall: ", kids_recalls_SVCrbf)
print("Precision: ", kids_precisions_SVCrbf)
print("F1 score: ", kids_f1_SVCrbf)
```

```
Accuracy:  [0.837605085093295]
Recall:  [0.6691488741990224]
Precision:  [0.7837857074262268]
F1 score:  [0.6990737047323206]
```

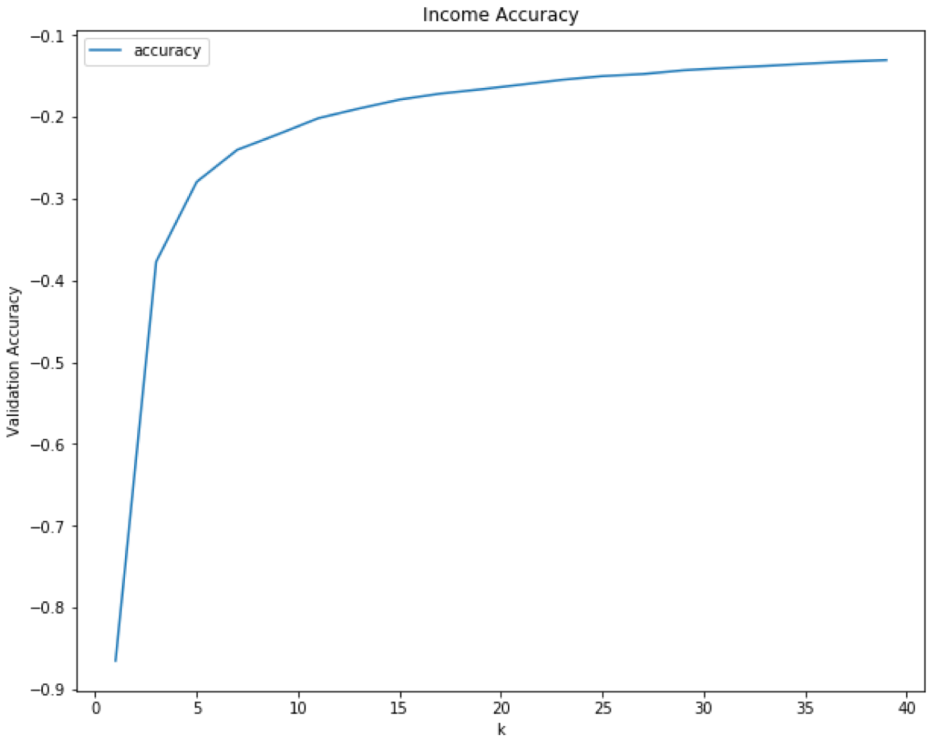
```
plt.figure(figsize=(16, 6))
plt.subplot(121)
plt.hist(y_predict_SVClin, bins=5, alpha=0.4)
plt.title("Kids prediction SVC linear")
plt.subplot(122)
plt.hist(y_predict_SVC_rbf, bins=5, alpha=0.4)
plt.title("Kids prediction SVC rbf")
plt.show()
```



Regression - Income

The regression here is based on the KNN regressor. Input features are only the essay text length and average word length.

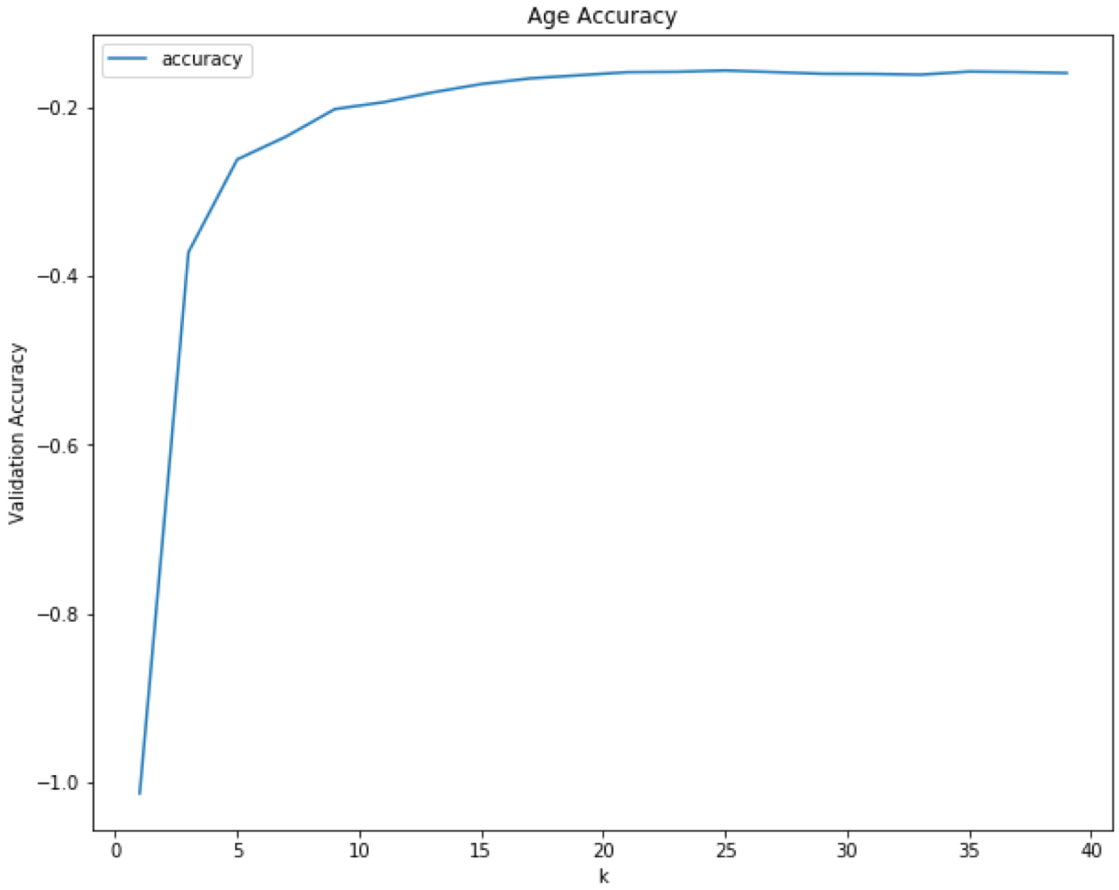
Accuracy is very poor and even negative.



Regression - Income

The regression here is based on the KNN regressor. Input features are only the i_me_frequency in the essays.

Accuracy is very poor and even negative.



Linear Regression - Income

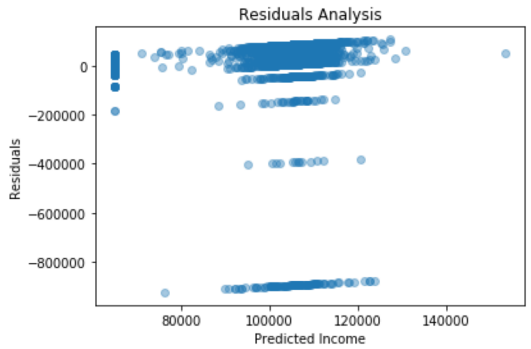
This regression is based on the LinearRegression from sklearn.linear_model

Again, both train and test accuracy score is very low and we observe a significant residual in the plot.

```
1  y_test_income, y_predict_income = linregr(income_prediction_features, income_prediction_labels)
```

```
Train score:
0.0021804643191750683
Test score:
0.001183747903015786
```

```
1  residuals = y_predict_income - y_test_income
2
3  plt.scatter(y_predict_income, residuals, alpha=0.4)
4  plt.title('Residuals Analysis')
5  plt.xlabel("Predicted Income")
6  plt.ylabel("Residuals")
7  plt.show()
```



Linear Regression - Age

This regression is based on the LinearRegression from sklearn.linear_model

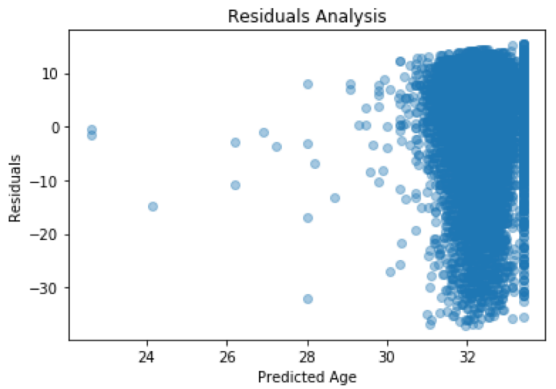
Again, both train and test accuracy score is very low and we observe a significant residual in the plot.

We observe that predicted age is nearly always between 30 and 34 years. (Even with a sharp cut-off at 34) Residuals (predicted - actual) is between -40 (actual age much higher than predicted) and 15 (actual age lower than predicted).

```
1  y_test_age, y_predict_age = linregr(age_prediction_features, age_prediction_labels)
```

```
Train score:
0.003647532104498641
Test score:
0.0034674981650129055
```

```
1  residuals = y_predict_age - y_test_age
2
3  plt.scatter(y_predict_age, residuals, alpha=0.4)
4  plt.title('Residuals Analysis')
5  plt.xlabel("Predicted Age")
6  plt.ylabel("Residuals")
7  plt.show()
```



KNN Classification of Age Segment

I wanted to see if using eight age segments as defined below in combination with the knn classifier and all the numerical features would yield improved accuracy.

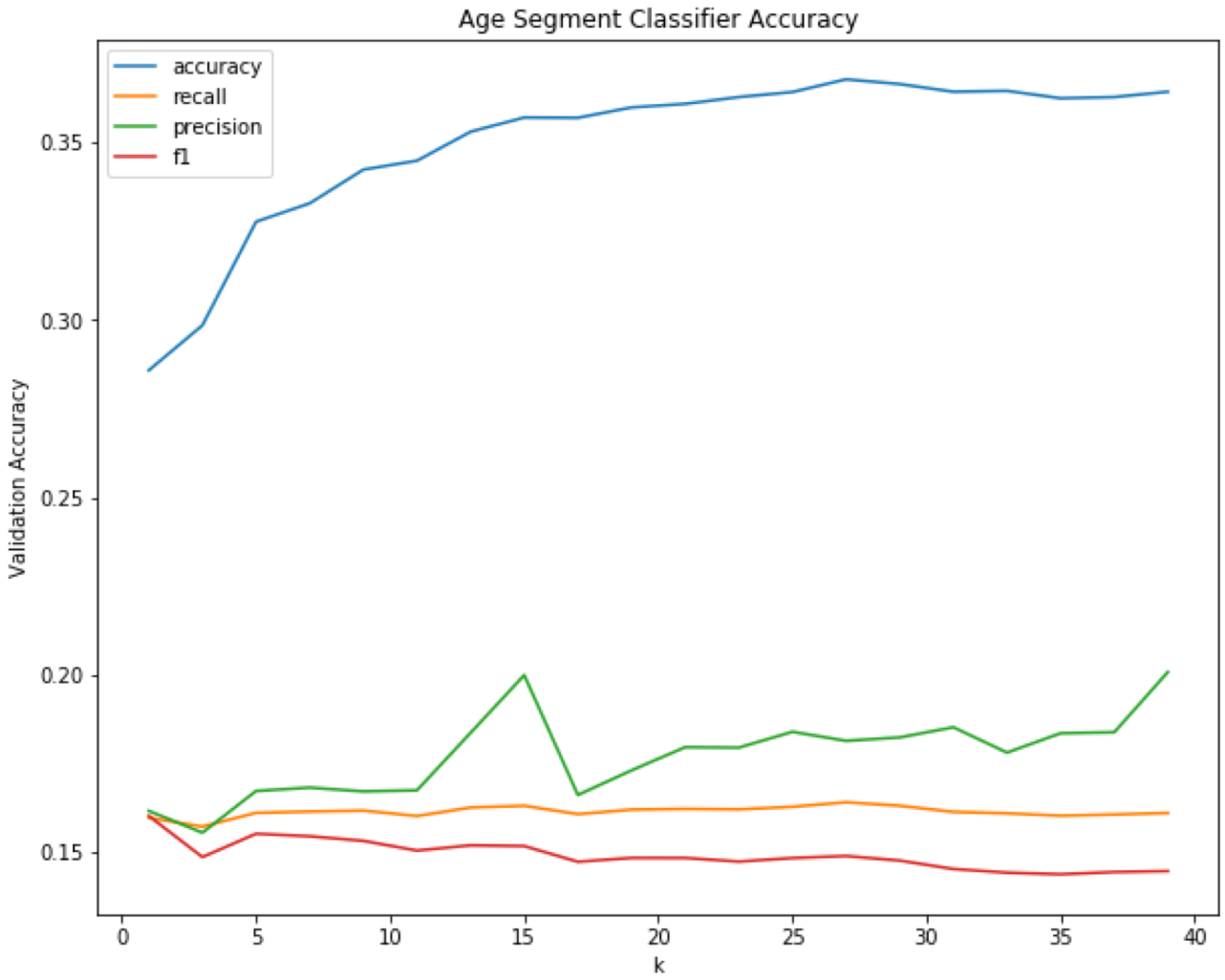
The accuracy is clearly improved compared to the regression methods, but still not great.

```
df['age_segment'] = pd.cut(df['age'], [0, 18, 23, 29, 39, 49, 59, 69, 120], labels=['0-18',
                                                                                     '19-23',
                                                                                     '24-29',
                                                                                     '30-39',
                                                                                     '40-49',
                                                                                     '50-59',
                                                                                     '60-69',
                                                                                     '70 and above'])

#print(df.age_segment.value_counts())

age_segment_map = {
    "0-18": 0,
    "19-23": 1,
    "24-29": 2,
    "30-39": 3,
    "40-49": 4,
    "50-59": 5,
    "60-69": 6,
    "70 and above": 7
}

df["age_segment_code"] = df.age_segment.map(age_segment_map)
```



Conclusion

The knn classifier fails to predict the zodiac sign of the user. Random guessing would be as accurate.

The knn classifier has the ability to predict if the user has kids or not with good accuracy.

The knn classifier has good ability to classify if a user is a smoker or not if the smoker features are merged into two classes only. For the original feature classification, the classifier will always predict non-smoker with low precision and recall.

The SVC classifier predicts if the user has kids with same accuracy as knn. However, if many features are included, SVC is very slow in the training phase.

Using regression techniques like knn and LinearRegression fails to predict income and age based on the essay length, average word length and I/me frequency.

By segmenting the age into eight age segments and using knn classifier with most features included, age can be predicted with about 36% accuracy.

The possibilities of testing different approaches to improve the predictions are almost endless. More careful selection of the features that go into the training and different feature segmentation would possibly give better metrics.