



The least correlated features are age, job, marital status, education, month, and day. so we going to drop them.

```
In [44]: # trimming our data set
bank.drop(['age','job','marital','education','month','day'],axis=1, inplace=True)
```

Here we define the target 'y' which contains whether client subscribed a term deposit or not.

```
In [45]: cols = bank.columns
target_col = 'y'
feat_cols = [c for c in cols if c != target_col]

# there is nothing magical about the X and y notation here.
# however, it seems to be a fairly standard notation, so we will use it here
X = bank[feat_cols].values
y = bank[target_col].values
```

We split the data to train and test sets with ratio of 80% 20%

```
In [46]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

We fit the train set to a KNN classifier model with 2 cluster.

```
In [47]: # define and fit our model
model = KNeighborsClassifier(n_neighbors=2, n_jobs=-1)
model.fit(X_train, y_train)
```

```
Out[47]: KNeighborsClassifier(n_jobs=-1, n_neighbors=2)
```

```
In [48]: # gather the predictions that our model made for our test set
preds = model.predict(X_test)

# display the actuals and predictions for the test set
print('Actuals for test data set')
print(y_test)
print('Predictions for test data set')
print(preds)
```

Actuals for test data set
[0 0 ... 0 0 0]
Predictions for test data set
[0 0 ... 0 0 0]

```
In [49]: #compare the two sets for 'subscribed'
differs = y_test - preds
print('Differences between the two sets')
print(differs)
```

Differences between the two sets
[0 0 ... 0 0 0]

```
In [50]: # R^2 (coefficient of determination) regression score function.
# Best possible score is 1.0 and it can be negative
# (because the model can be arbitrarily worse).
# A constant model that always predicts the expected value of y,
# disregarding the input features, would get a R^2 score of 0.0.
from sklearn.metrics import r2_score
print(r2_score(y_test, preds))
```

-0.17576494715312996

```
In [51]: # Explained variance regression score function
# Best possible score is 1.0, lower values are worse.
from sklearn.metrics import explained_variance_score
print(explained_variance_score(y_test, preds))
```

-0.12024899165847525

The accuracy is ~17%. the model is not performing well.

```
In [52]: scores = []
print(f'Features: {feat_cols} \nTarget: {target_col}')

# remember the ending number for range is not inclusive
for k in range(2, 20):
    # output to let us know where we are
    print(f'Evaluating {k} clusters')

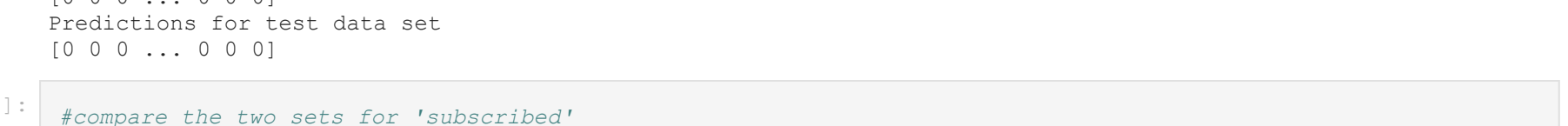
    # n_jobs=-1 will use all processors on your system
    model = KNeighborsRegressor(n_neighbors=k, n_jobs=-1)
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))
```

Features: ['default', 'balance', 'housing', 'loan', 'contact', 'duration', 'campaign', 'pdays', 'previous', 'poutcome']
Target: y
Evaluating 2 clusters
Evaluating 3 clusters
Evaluating 4 clusters
Evaluating 5 clusters
Evaluating 6 clusters
Evaluating 7 clusters
Evaluating 8 clusters
Evaluating 9 clusters
Evaluating 10 clusters
Evaluating 11 clusters
Evaluating 12 clusters
Evaluating 13 clusters
Evaluating 14 clusters
Evaluating 15 clusters
Evaluating 16 clusters
Evaluating 17 clusters
Evaluating 18 clusters
Evaluating 19 clusters

```
In [53]: scores
```

Out[53]: [-0.127556449963022897, -0.022771892071443034, -0.04026101499003165, 0.08323819359788398, 0.05040704565665444, 0.12406872867049445, 0.13896716000632552, 0.14989702980451117, 0.15714956212988263, 0.16007682818566893, 0.16648398482003162, 0.17177995090769793, 0.1768347569979308, 0.178729574206205, 0.1846746321689379, 0.1901882498113637, 0.18923267805279098, 0.19315293150546287]

```
In [54]: # display the results
plt.plot(range(2, 20), scores)
plt.scatter(range(2, 20), scores)
plt.grid()
_ = plt.xticks(range(2, 20))
```



17 clusters and 19 clusters are the highest performing model. We are going to pick 17 clusters because it is the smallest highest K number

```
In [55]: # define and fit our model
model = KNeighborsRegressor(n_neighbors=17, n_jobs=-1)
model.fit(X_train, y_train)

# gather the predictions that our model made for our test set
preds = model.predict(X_test)

# display the actuals and predictions for the test set
print('Actuals for test data set')
print(y_test)
print('Predictions for test data set')
print(preds)
```

Actuals for test data set
[0 0 ... 0 0 0]
Predictions for test data set
[0. 0.5882353 0.5882353 ... 0.5882353 0.5882353 0.]

```
In [56]: differs = y_test - preds

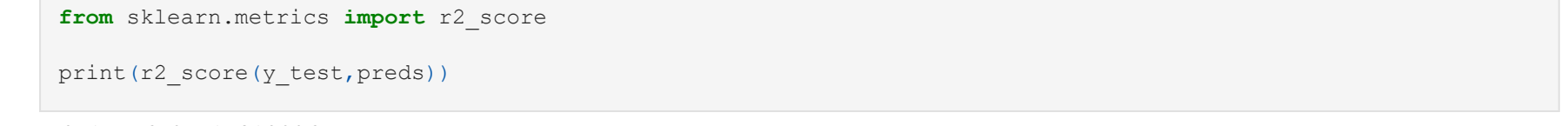
print(f'Differences between the two sets:\n{differs}')
print(f'r2_score: {r2_score(y_test, preds)}')
```

Differences between the two sets:
[0. -0.5882353 -0.5882353 ... -0.5882353 -0.5882353]
r2_score: 0.1901882498153637

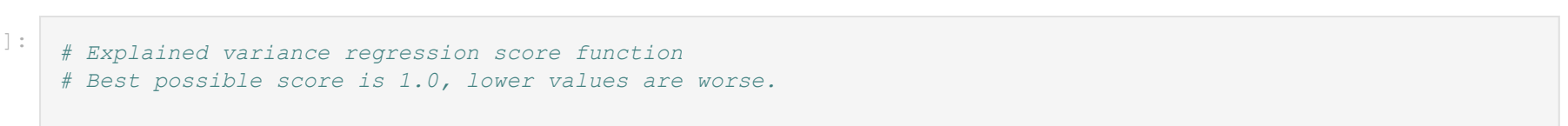
The model accuracy is 19% which is better than the first model but still not performing well.

```
In [57]: #visualization
pc = PCA()
tr_pca = pc.fit_transform(X_train)
te_pca = pc.transform(X_test)
```

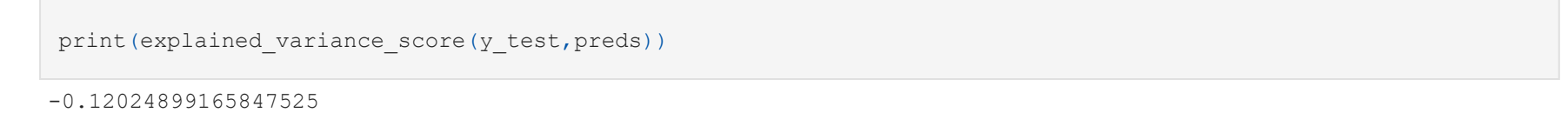
```
In [58]: # indexing of a matrix in Python (numpy) is (rows, cols)
_ = plt.scatter(tr_pca[:, 0], tr_pca[:, 1], c=y_train)
_ = plt.colorbar()
```



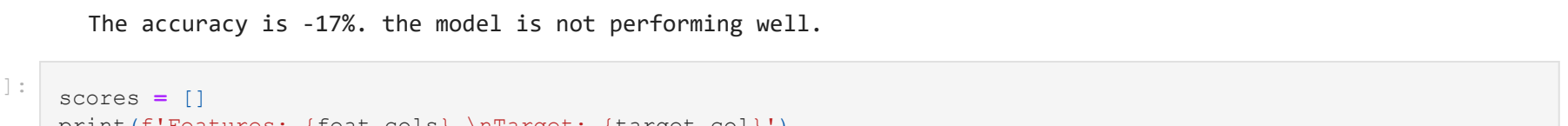
```
In [59]: _ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=y_test)
_ = plt.colorbar()
```



```
In [60]: _ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=preds)
_ = plt.colorbar()
```



```
In [61]: _ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=differs)
_ = plt.colorbar()
```



Clearly we need to improve the model's performance. To do that, we need to normalize data.

```
In [62]: # we only want to normalize our feature columns in the dataset and I don't want to resplit the dataset.
# So, I will normalize the X_train and X_test only
from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()
X_tr_norm = min_max_scaler.fit_transform(X_train)
X_te_norm = min_max_scaler.fit_transform(X_test)
```

```
In [63]: new_bank_tr = pd.DataFrame(X_tr_norm, columns=feat_cols)
new_bank_tr['y'] = y_train
new_bank_tr.head(10)
```

Out[63]:

	default	balance	housing	loan	contact	duration	campaign	pdays	previous	poutcome	y
0	0.0	0.080511	0.0	1.0	0.0	0.014640	0.000000	0.000000	0.000000	0.0	0
1	0.0	0.085650	0.0	1.0	0.0	0.054697	0.016129	0.000000	0.000000	0.0	0
2	0.0	0.017684	1.0	1.0	0.0	0.026434	0.048387	0.000000	0.000000	0.0	0
3	0.0	0.071723	0.0	1.0	0.0	0.076251	0.161290	0.000000	0.000000	0.0	0
4	0.0	0.104561	1.0	1.0	1.0	0.053680	0.016129	0.000000	0.000000	0.0	1
5	0.0	0.107911	1.0	1.0	0.0	0.025257	0.177419	0.000000	0.000000	0.0	0
6	0.0	0.080511	0.0	1.0	0.0	0.053070	0.080645	0.000000	0.000000	0.0	0
7	0.0	0.076417	0.0	0.0	0.0	0.043514	0.016129	0.000000	0.000000	0.0	0
8	0.0	0.077788	1.0	1.0	0.0	0.030297	0.000000	0.000000	0.000000	0.0	0
9	0.0	0.084787	1.0	1.0	1.0	0.069134	0.000000	0.105505	0.007273	1.0	1

```
In [64]: new_auto_te = pd.DataFrame(X_te_norm, columns=feat_cols)
new_auto_te['y'] = y_test
new_auto_te.head(10)
```

Out[64]:

	default	balance	housing	loan	contact	duration	campaign	pdays	previous	poutcome	y
0	0.0	0.032552	0.0	1.0	0.0	0.060320	0.000000	0.000000	0.000000	0.000000	0
1	0.0	0.062850	1.0	1.0	0.0	0.026076	0.018519	0.000000	0.000000	0.000000	0
2	0.0	0.032137	0.0	1.0	1.0	0.071002	0.000000	0.000000	0.000000	0.000000	0
3	0.0	0.044349	1.0	1.0	1.0	0.097707	0.000000	0.399763	0.017241	0.666667	0
4	0.0	0.028963	1.0	0.0	1.0	0.038014	0.018519	0.000000	0.000000	0.000000	0
5	0.0	0.038031	1.0	1.0	1.0	0.040842	0.037037	0.000000	0.000000	0.000000	0
6	0.0	0.037546	1.0	0.0	1.0	0.029846	0.092593	0.000000	0.000000	0.000000	0
7	0.0	0.028073	1.0	1.0	1.0	0.026076	0.055556	0.000000	0.000000	0.000000	0
8	0.0	0.026847	0.0	1.0	1.0	0.016023	0.000000	0.000000	0.000000	0.000000	0
9	0.0	0.029368	1.0	0.0	0.0	0.184417	0.018519	0.000000	0.000000	0.000000	0

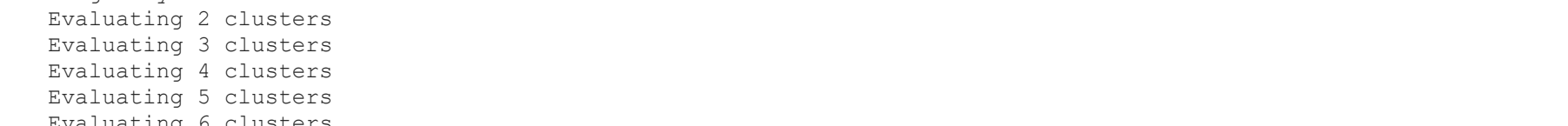
```
In [65]: scores_norm = []
print(f'Features: {feat_cols} \nTarget: {target_col}')

# remember the ending number for range is not inclusive
for k in range(2, 30):
    # output to let us know where we are
    print(f'Evaluating {k} clusters')

    # n_jobs=-1 will use all processors on your system
    model_norm = KNeighborsRegressor(n_neighbors=k, n_jobs=-1)
    model_norm.fit(X_tr_norm, y_train)
    scores_norm.append(model_norm.score(X_te_norm, y_test))
```

Features: ['default', 'balance', 'housing', 'loan', 'contact', 'duration', 'campaign', 'pdays', 'previous', 'poutcome']
Target: y
Evaluating 2 clusters
Evaluating 3 clusters
Evaluating 4 clusters
Evaluating 5 clusters
Evaluating 6 clusters
Evaluating 7 clusters
Evaluating 8 clusters
Evaluating 9 clusters
Evaluating 10 clusters
Evaluating 11 clusters
Evaluating 12 clusters
Evaluating 13 clusters
Evaluating 14 clusters
Evaluating 15 clusters
Evaluating 16 clusters
Evaluating 17 clusters
Evaluating 18 clusters
Evaluating 19 clusters
Evaluating 20 clusters
Evaluating 21 clusters
Evaluating 22 clusters
Evaluating 23 clusters
Evaluating 24 clusters
Evaluating 25 clusters
Evaluating 26 clusters
Evaluating 27 clusters
Evaluating 28 clusters
Evaluating 29 clusters

```
In [66]: # display the results
plt.plot(range(2, 30), scores_norm)
plt.scatter(range(2, 30), scores_norm)
plt.grid()
_ = plt.xticks(range(2, 30))
```



Seems like k=17 is the optimal

```
In [67]: # define and fit our model with k=17
model_norm = KNeighborsRegressor(n_neighbors=17, n_jobs=-1)
model_norm.fit(X_tr_norm, y_train)

# gather the predictions that our model made for our test set
preds_norm = model_norm.predict(X_te_norm)
```

```
# display the actuals and predictions for the test set
print('Actuals for test data set')
print(y_test)
print('Predictions for test data set')
print(preds_norm)
```

Actuals for test data set
[0 0 ... 0 0 0]
Predictions for test data set
[0. 0. -0.0952381 ... 0. 0. 0.]

```
In [68]: differs_norm = y_test - preds_norm

print(f'Differences between the two sets:\n{differs_norm}')
print(f'r2_score: {r2_score(y_test, preds_norm)}')
```

Differences between the two sets:
[0. -0.0952381 ... 0. 0. 0.]
r2_score: 0.19855738333203994

The model performance is slightly improved by 8.088 which it doesn't worth to normalize the data.

Conclusion

In sum, both exercise and project models are performing far below the acceptable model. This performance is because many features are not strongly correlated with the target. In order to improve our models, we need to go back and do better with feature selection.

References

Harrison, O. (2018, Sep 18). Machine Learning Basics with the K-Nearest Neighbors Algorithm. Retrieved from Toward data science: <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>

Wijaya, C. Y. (2020, Jul 12). Jupyter Notebook to PDF in a few lines. Retrieved from Toward data science: <https://towardsdatascience.com/jupyter-notebook-to-pdf-in-a-few-lines-3c48d68a7a63>