Assignment for Week 2 - KNN

Task 1 - KNN Exercise

Data Set: https://archive.ics.uci.edu/ml/datasets/Heart+Disease
Notice, this is the same dataset that you worked with last week. Feel free to use your cleaned up data file or you can use the one that I have posted in WorldClass.

Objective: Predict heart disease in patients.

- Get to know your data, start out by data exploration. Summarized your finding.
- Divide the data into training set and test set randomly with ratio 80:20. Make prediction based on 1-nearest neighbor. What is the error rate of this approach? Report your results in a confusion matrix.
- Use different values for K, what is the optimal value of K from your experiments? Report the error rate of the optimal K value and its confusion matrix. Is there any improvement (by how much) over 1-nearest neighbor?
- Is there anything else you can do to improve your model? If yes, demonstrate your approach. (Hint: there is always something that you can try, unless your accuracy score is 100%)

Task 2 - KNN Project

Data Set: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

Objective: Determine whether the product is 'yes' or 'no' subscription

- Use KNN to provide an analyze of the given data set. Make sure you are answering the stated objective above.
- What is your optimal K? What is the accuracy rate?
- Discover any insights from this analysis? (include numbers/graphs corresponding to your reasoning)

Deliverables:

Upload your notebook's .ipynb file (This assignment can be done on 1 notebook or 2. Your preference.) Please include the jupyter notebook file and a pdf printout of your notebook.

Important: Make sure your provide complete and thorough explanations for all of your analysis. You need to defend your thought processes and reasoning.

Introduction

```
Week 2 is about KKN meethod. KNN is a supervised machine learning algorithm that can be used to solve
classification and regression problems. This week assignment has two part. part 1 is exercise on Heart
disease data from week 1. the second part is the project where we use KNN on Bank data.
```

Dataset

```
For the exercise, we are going to use Heart disease dataset which is set of records of people with there
diagnosis of heart disease. The data contains 15 variables where 'num' is the target and the other 14
are our features. those features are various from chest pain to number of years of smoking. As for the
project, the dataset is Bank dataset which contains personal information such as marital status and
education level. This dataset contains 17 columns where 'Y' is the target. Because this data has many
categorical variable, we need to convert them to numeric in order to fit them in KNN model.
```

exercise 1

```
Loading required libraries
```

In [53]:
```
# pandas
import pandas as pd

# sklearn
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA

# plotting
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set()
```

```
Loading cleaned Heart disease dataset.
```

In [2]:
```
# loading heart disease data
heartd = pd.read_csv('heart.disease.data.clean.csv')
```

```
Data is clean and numeric. No cleaning needed except for dropping unnecessary columns for KNN model
which will be determined after seeing correlation.
```

In [3]:
```
heartd.head(10)
```

Out[3]:

	age	sex	cp	trestbps	chol	cigs	years	fbs	famhist	restecg	thalach	exang	thal	num
0	63	1	1	145	233	50.0	20.0	1	1	2	150	0	6	0
1	67	1	4	160	286	40.0	40.0	0	1	2	108	1	3	2
2	67	1	4	120	229	20.0	35.0	0	1	2	129	1	7	1
3	37	1	3	130	250	0.0	0.0	0	1	0	187	0	3	0
4	41	0	2	130	204	0.0	0.0	0	1	2	172	0	3	0
5	56	1	2	120	236	20.0	20.0	0	1	0	178	0	3	0
6	62	0	4	140	268	0.0	0.0	0	1	2	160	0	3	3
7	57	0	4	120	354	0.0	0.0	0	1	0	163	1	3	0
8	63	1	4	130	254	0.0	0.0	0	0	2	147	0	7	2
9	53	1	4	140	203	20.0	25.0	1	1	2	155	1	7	1

In [4]:
```
heartd.info()
```

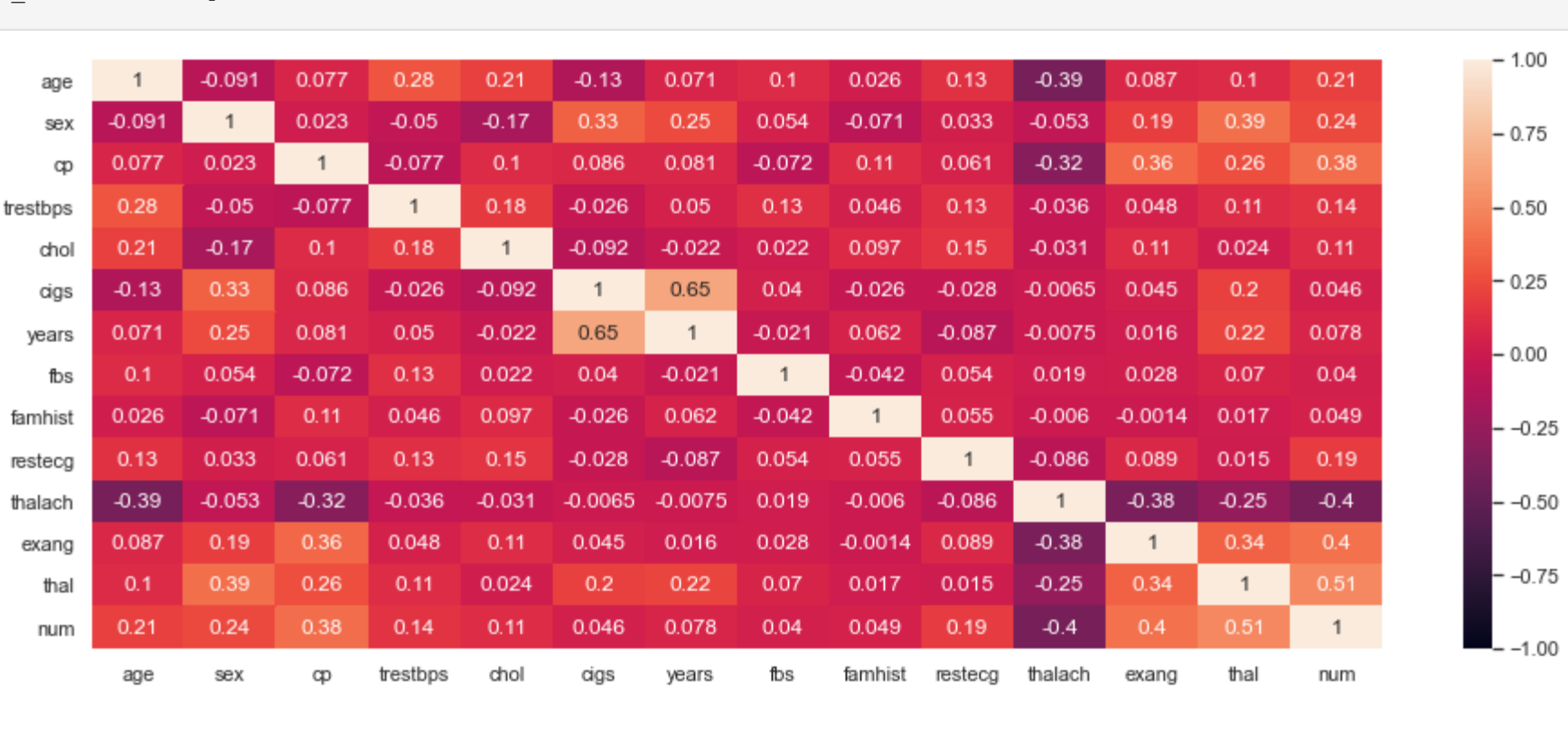
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 282 entries, 0 to 281
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       282 non-null    int64
 1   sex       282 non-null    int64
 2   cp        282 non-null    int64
 3   trestbps  282 non-null    int64
 4   chol      282 non-null    int64
 5   cigs      282 non-null    float64
 6   years     282 non-null    float64
 7   fbs       282 non-null    int64
 8   famhist   282 non-null    int64
 9   restecg   282 non-null    int64
 10  thalach   282 non-null    int64
 11  exang     282 non-null    int64
 12  thal      282 non-null    int64
 13  num       282 non-null    int64
dtypes: float64(2), int64(12)
memory usage: 31.0 KB
```

In [5]:
```
heartd.describe()
```

Out[5]:

	age	sex	cp	trestbps	chol	cigs	years	fbs	famhist	restecg	thalach	
count	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282
mean	54.411348	0.677305	3.163121	131.195035	247.705674	16.836011	15.347364	0.148936	0.620567	1.014184	149.826241	0
std	9.053083	0.468338	0.955405	16.739821	46.178771	18.876755	15.276814	0.356658	0.486108	0.998118	22.737594	0
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	0.000000	0.000000	0.000000	88.000000	0
25%	48.000000	0.000000	3.000000	120.000000	213.000000	0.000000	0.000000	0.000000	0.000000	0.000000	133.250000	0
50%	55.000000	1.000000	3.000000	130.000000	244.000000	11.976385	15.000000	0.000000	1.000000	2.000000	153.500000	0
75%	61.000000	1.000000	4.000000	140.000000	277.000000	30.000000	30.000000	0.000000	1.000000	2.000000	165.750000	1
max	77.000000	1.000000	4.000000	170.000000	360.000000	75.000000	54.000000	1.000000	1.000000	2.000000	202.000000	1

In [7]:
```
plt.figure(figsize=(16, 6))
_ = sns.heatmap(heartd.corr(), vmin=-1, vmax=1, annot=True)
```




```
Since maximum correlation with num is 0.51, i am going to drop very low correlated variables. In this
case, cigarrets consumption and number of years of smoking are below 0.07. Also fbs is 0.04 and family
history is 0.049. those variables have very low correlation with the target compared to other
independent variables which most of them have low correlation with num. However, I cannot drop most of
them and only have 1 or 2 variables.
```

In [9]:
```
# trimming our data set
# cigs, years, fbs, and famhist
heartd.drop(['cigs','years','fbs','famhist' ],axis=1, inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-9-f52be1cd9623> in <module>
      1 # trimming our data set
      2 # cigs, years, fbs, and famhist
----> 3 heartd.drop(['cigs','years','fbs','famhist' ],axis=1, inplace=True)

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in drop(self, labels, axis, index, columns, lev
el, inplace, errors)
   4306             weight  1.0     0.8
   4307         """
-> 4308         return super().drop(
   4309             labels=labels,
   4310             axis=axis,

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in drop(self, labels, axis, index, columns, l
evel, inplace, errors)
   4151         for axis, labels in axes.items():
   4152             if labels is not None:
-> 4153                 obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4154
   4155         if inplace:

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in _drop_axis(self, labels, axis, level, erro
rs)
   4186                 new_axis = axis.drop(labels, level=level, errors=errors)
   4187             else:
-> 4188                 new_axis = axis.drop(labels, errors=errors)
   4189             result = self.reindex(**{axis_name: new_axis})
   4190

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels, errors)
   5589         if mask.any():
   5590             if errors != "ignore":
-> 5591                 raise KeyError(f"{labels[mask]} not found in axis")
   5592         indexer = indexer[~mask]
   5593         return self.delete(indexer)

KeyError: "['cigs' 'years' 'fbs' 'famhist'] not found in axis"
```

In [10]:
```
heartd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 282 entries, 0 to 281
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       282 non-null    int64
 1   sex       282 non-null    int64
 2   cp        282 non-null    int64
 3   trestbps  282 non-null    int64
 4   chol      282 non-null    int64
 5   restecg   282 non-null    int64
 6   thalach   282 non-null    int64
 7   exang     282 non-null    int64
 8   thal      282 non-null    int64
 9   num       282 non-null    int64
dtypes: int64(10)
memory usage: 22.2 KB
```

In [11]:
```
# Defining our features and targets
cols = heartd.columns
target_col = 'num'
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 is here
X = heartd[feat_cols].values
y = heartd[target_col].values
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [13]:
```
# Make prediction based on 1-nearest neighbor.
#What is the error rate of this approach?
#Report your results in a confusion matrix.


# define and fit our model
model = KNeighborsRegressor(n_neighbors=1, n_jobs=-1)
model.fit(X_train, y_train)
```

Out[13]:
```
KNeighborsRegressor(n_jobs=-1, n_neighbors=1)
```

In [14]:
```
# gather the predictations 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 1 3 0 0 1 1 4 3 1 0 3 0 4 0 4 0 0 0 1 2 0 0 0 0 3 0 2 0 2 0 3 0 0 2
 3 0 1 4 4 1 0 0 0 0 3 0 0 3 1 0 1 2 0 3]
Predictions for test data set
[2. 1. 0. 0. 1. 0. 3. 0. 3. 4. 2. 0. 2. 0. 0. 2. 1. 1. 0. 1. 3. 0. 1. 0.
 0. 0. 0. 2. 0. 0. 1. 3. 0. 2. 0. 0. 1. 4. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 3. 0. 1.]
```

In [15]:
```
#compare the two sets
differs = y_test - preds
print('Differences between the two sets')
print(differs)
```

```
Differences between the two sets
[-2. -1.  0.  1.  2.  0. -3.  1. -2.  0.  1.  1. -2.  3.  0.  2. -1.  3.
  0. -1. -3.  1.  1.  0.  0.  0.  1.  0.  2. -1. -1.  0.  1.  0.  0.
  1. -1.  0.  1.  4.  4.  1. -1.  0.  0.  0.  3. -1.  0.  3.  1.  0.  1.
 -1.  0.  2.]
```

In [16]:
```
# 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))
```

```
-0.1974050046339202
```

In [17]:
```
# 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))
```

```
-0.13561940067964162
```

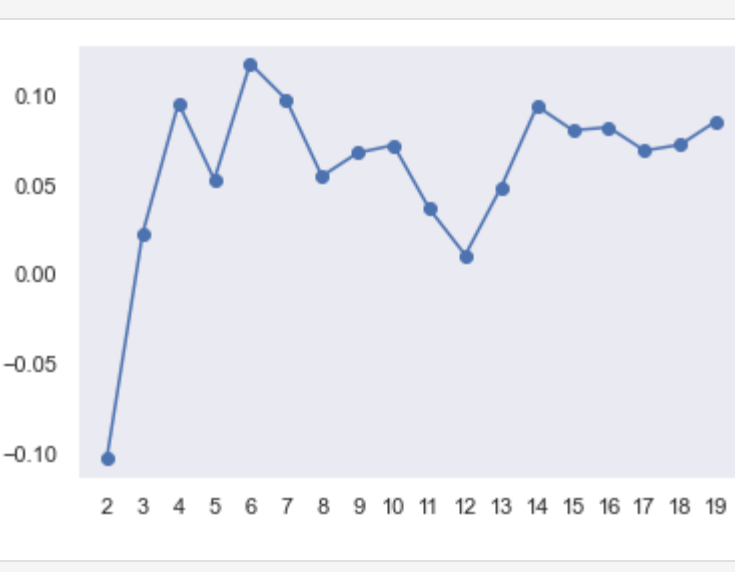
In [18]:
```
# Determining the optimal number of clusters
scores = []
print(f'Features: {feat_cols} \nTarget: {target_col}')

# 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: ['age', 'sex', 'cp', 'trestbps', 'chol', 'restecg', 'thalach', 'exang', 'thal']
Target: num
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 [19]:
```
scores
```

Out[19]:
```
[-0.1027571825764595,
 0.02270620945319768,
 0.0953429101019464,
 0.05264133456904563,
 0.11735403151065815,
 0.09745418093094527,
 0.05462233549582962,
 0.06759802743738497,
 0.07192307692307709,
 0.03638584854357052,
 0.010722376686232216,
 0.04849438719831545,
 0.09359100452043656,
 0.08015034496962226,
 0.08182670586190932,
 0.06883055244667802,
 0.07219047128685685,
 0.08526413345690453]
```

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



In [22]:
```
# define and fit our model
model = KNeighborsRegressor(n_neighbors=6, n_jobs=-1)
model.fit(X_train, y_train)

# gather the predictations 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)
```

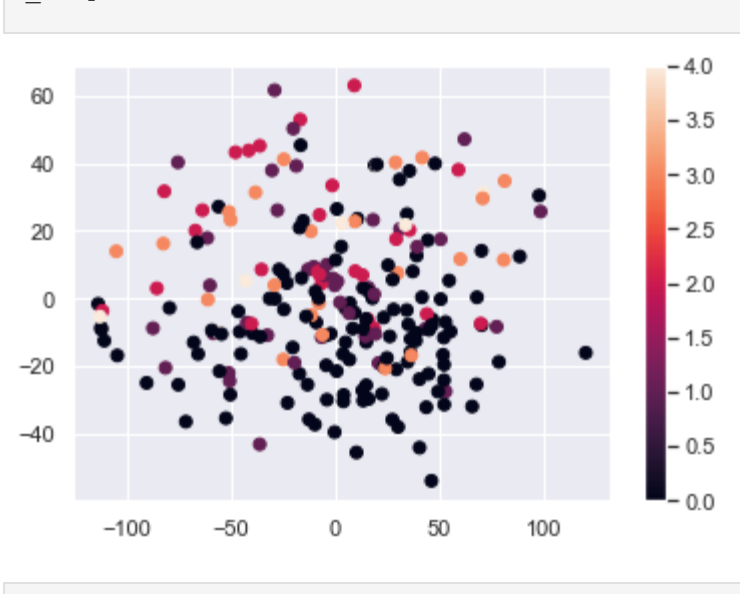
```
Actuals for test data set
[0 0 0 1 3 0 0 1 1 4 3 1 0 3 0 4 0 4 0 0 0 1 2 0 0 0 0 3 0 2 0 2 0 3 0 0 2
 3 0 1 4 4 1 0 0 0 0 3 0 0 3 1 0 1 2 0 3]
Predictions for test data set
[1.16666667 0.33333333 0.         0.33333333 1.         1.16666667
 1.         0.5        1.83333333 2.         1.83333333 0.
 1.16666667 0.33333333 1.         1.         1.16666667 2.33333333
 0.66666667 2.         1.83333333 1.16666667 1.33333333 0.5
 0.83333333 0.16666667 0.         1.         1.         1.16666667
 0.16666667 2.16666667 0.66666667 1.5        0.         0.33333333
 0.33333333 1.83333333 0.83333333 0.83333333 1.33333333 0.
 0.33333333 0.66666667 0.         0.16666667 0.33333333 1.5
 0.66666667 0.66666667 0.5        1.16666667 0.5        0.
 0.83333333 0.66666667 1.33333333]
```

In [23]:
```
differs = y_test - preds

print(f'Differences between the two sets:\n{differs}')

print(f'r2_score: {r2_score(y_test,preds)}')
```

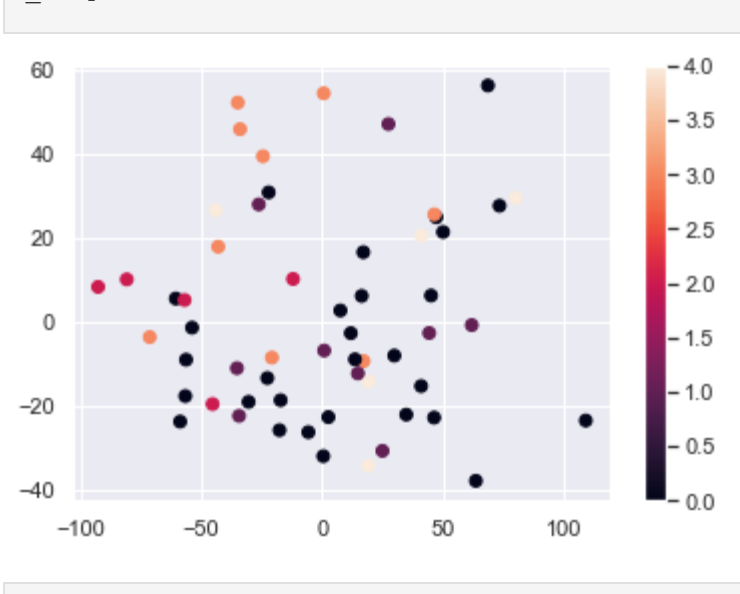
```
Differences between the two sets:
[-1.16666667 -0.33333333  0.          0.66666667  2.         -1.16666667
 -1.          0.5        -0.83333333  2.          1.16666667  1.
 -1.16666667  2.66666667 -1.          3.         -1.16666667  1.66666667
 -0.66666667 -2.         -1.83333333 -0.16666667  0.66666667 -0.5
 -0.83333333 -0.16666667  0.          2.         -1.          0.83333333
 -0.16666667 -0.16666667 -0.66666667  1.5         0.         -0.33333333
  1.66666667  1.16666667 -0.83333333  0.16666667  2.66666667  4.
  0.66666667 -0.66666667  0.         -0.16666667 -0.33333333  1.5
 -0.66666667 -0.66666667  2.5        -0.16666667 -0.5         1.
  1.16666667 -0.66666667  1.66666667]
r2_score: 0.11735403151065815
```

In [24]:
```
# visualize model
pc = PCA()
tr_pca = pc.fit_transform(X_train)
te_pca = pc.transform(X_test)
```

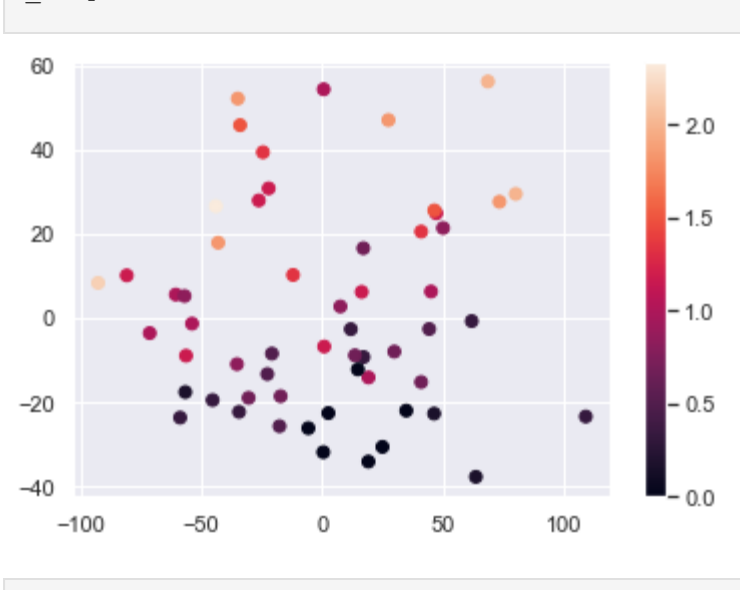
In [26]:
```
# indexing of a matrix in Python (numpy) is [rows, cols]
_ = plt.scatter(tr_pca[:, 0], tr_pca[:, 1], c=y_train)
_ = plt.colorbar()
```



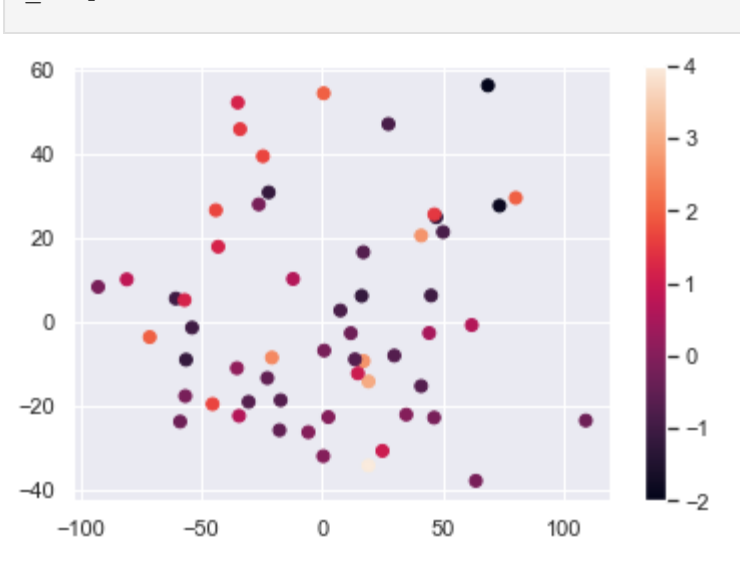
In [27]:
```
_ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=y_test)
_ = plt.colorbar()
```



In [28]:
```
_ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=preds)
_ = plt.colorbar()
```



In [29]:
```
_ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=differs)
_ = plt.colorbar()
```



In [30]:
```
# normalize data to improve model maybe
heartd.describe()
```

Out[30]:

	age	sex	cp	trestbps	chol	restecg	thalach	exang	thal	num
count	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000	282.000000
mean	54.411348	0.677305	3.163121	131.195035	247.705674	1.014184	149.826241	0.326241	4.677305	0.907801
std	9.053083	0.468338	0.955405	16.739821	46.178771	0.998118	22.737594	0.469670	1.936386	1.224894
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	88.000000	0.000000	3.000000	0.000000
25%	48.000000	0.000000	3.000000	120.000000	213.000000	0.000000	133.250000	0.000000	3.000000	0.000000
50%	55.000000	1.000000	3.000000	130.000000	244.000000	2.000000	153.500000	0.000000	3.000000	0.000000
75%	61.000000	1.000000	4.000000	140.000000	277.000000	2.000000	165.750000	1.000000	7.000000	2.000000
max	77.000000	1.000000	4.000000	170.000000	360.000000	2.000000	202.000000	1.000000	7.000000	4.000000

In [32]:
```
heartd.head(10)
```

Out[32]:

	age	sex	cp	trestbps	chol	restecg	thalach	exang	thal	num
0	63	1	1	145	233	2	150	0	6	0
1	67	1	4	160	286	2	108	1	3	2
2	67	1	4	120	229	2	129	1	7	1
3	37	1	3	130	250	0	187	0	3	0
4	41	0	2	130	204	2	172	0	3	0
5	56	1	2	120	236	0	178	0	3	0
6	62	0	4	140	268	2	160	0	3	3
7	57	0	4	120	354	0	163	1	3	0
8	63	1	4	130	254	2	147	0	7	2
9	53	1	4	140	203	2	155	1	7	1

In [33]:
```
# 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 [34]:
```
new_heartd_tr = pd.DataFrame(X_tr_norm,columns=feat_cols)
new_heartd_tr['num'] = y_train
new_heartd_tr.head(10)
```

Out[34]:

	age	sex	cp	trestbps	chol	restecg	thalach	exang	thal	num
0	0.458333	1.0	0.666667	0.407895	0.508547	1.0	0.684211	0.0	0.00	0
1	0.687500	0.0	1.000000	0.578947	0.717949	0.0	0.157895	0.0	0.00	2
2	0.416667	1.0	0.333333	0.473684	0.598291	0.0	0.728070	0.0	0.00	0
3	0.812500	1.0	0.666667	0.315789	0.645299	0.0	0.552632	0.0	1.00	0
4	0.625000	1.0	0.333333	0.605263	0.405983	0.0	0.666667	1.0	0.00	0
5	0.583333	1.0	0.333333	0.394737	0.576923	0.0	0.464912	0.0	1.00	1
6	0.562500	1.0	0.333333	0.342105	0.470085	0.0	0.789474	0.0	0.00	0
7	0.479167	1.0	0.333333	0.342105	0.850427	0.0	0.736842	0.0	0.00	0
8	0.583333	1.0	0.666667	0.736842	0.179487	0.0	0.754386	0.0	0.00	0
9	0.583333	1.0	1.000000	0.605263	0.282051	0.0	0.526316	0.0	0.75	0

In [35]:
```
new_heartd_te = pd.DataFrame(X_te_norm,columns=feat_cols)
new_heartd_te['mpg'] = y_test
new_heartd_te.head(10)
```

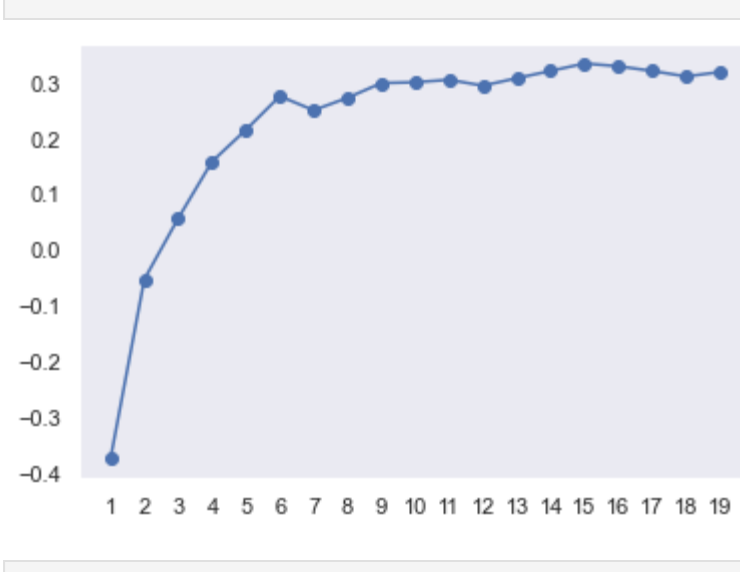
Out[35]:

	age	sex	cp	trestbps	chol	restecg	thalach	exang	thal	mpg
0	0.675676	1.0	0.666667	0.384615	0.450	0.0	0.536842	0.0	1.0	0
1	0.189189	0.0	0.666667	0.046154	0.000	0.0	0.842105	0.0	0.0	0
2	0.432432	1.0	0.666667	0.384615	0.525	1.0	0.821053	0.0	0.0	0
3	0.756757	1.0	0.000000	0.507692	0.705	1.0	0.831579	0.0	0.0	1
4	0.567568	1.0	0.333333	0.476923	0.890	1.0	0.600000	0.0	0.0	3
5	1.000000	0.0	0.333333	0.230769	0.640	1.0	0.273684	1.0	0.0	0
6	0.378378	0.0	0.666667	0.538462	0.835	1.0	0.494737	0.0	0.0	0
7	0.432432	1.0	1.000000	0.538462	0.310	1.0	0.631579	1.0	1.0	1
8	0.567568	1.0	1.000000	0.630769	0.385	0.0	0.105263	0.0	1.0	1
9	0.648649	1.0	1.000000	0.507692	0.125	1.0	0.315789	1.0	0.0	4

In [36]:
```
scores_norm = []
print(f'Features: {feat_cols} \nTarget: {target_col}')

# remember the ending number for range is not inclusive
for k in range(1, 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_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: ['age', 'sex', 'cp', 'trestbps', 'chol', 'restecg', 'thalach', 'exang', 'thal']
Target: num
Evaluating 1 clusters
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 [37]:
```
# display the resutls
plt.plot(range(1, 20), scores_norm)
plt.scatter(range(1, 20), scores_norm)
plt.grid()
_ =plt.xticks(range(1, 20))
```



In [38]:
```
# define and fit our model with k=12
model_norm = KNeighborsRegressor(n_neighbors=15, n_jobs=-1)
model_norm.fit(X_tr_norm, y_train)

# gather the predictations 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 1 3 0 0 1 1 4 3 1 0 3 0 4 0 4 0 0 0 1 2 0 0 0 0 3 0 2 0 2 0 3 0 0 2
 3 0 1 4 4 1 0 0 0 0 3 0 0 3 1 0 1 2 0 3]
Predictions for test data set
[1.13333333 0.         0.2        0.13333333 0.06666667 0.2
 0.13333333 1.86666667 1.26666667 1.6        1.73333333 1.2
 1.73333333 1.66666667 0.2        1.6        0.13333333 1.93333333
 1.2        0.13333333 0.33333333 1.8        1.66666667 0.06666667
 0.13333333 0.         0.13333333 1.73333333 0.26666667 2.
 0.4        2.06666667 1.6        1.53333333 0.         0.2
 1.66666667 2.2        0.         0.2        1.46666667 0.93333333
 0.2        0.06666667 0.13333333 0.         0.2        1.73333333
 1.73333333 0.         0.13333333 1.73333333 0.         0.86666667
 1.66666667 0.         1.8       ]
```

In [39]:
```
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:
[-1.13333333  0.         -0.2         0.86666667  2.93333333 -0.2
 -0.13333333 -0.86666667 -0.26666667  2.4         1.26666667 -0.2
 -1.73333333  1.33333333 -0.2         2.4        -0.13333333  2.06666667
 -1.2        -0.13333333 -0.33333333 -0.8         0.33333333 -0.06666667
 -0.13333333  0.         -0.13333333  1.26666667 -0.26666667  0.
 -0.4        -0.06666667 -1.6         1.46666667  0.         -0.2
  0.33333333  0.8         0.          0.8         2.53333333  3.06666667
  0.8        -0.06666667 -0.13333333  0.         -0.2         1.26666667
 -1.73333333  0.          2.86666667 -0.73333333  0.          0.13333333
  0.33333333  0.          1.2       ]
r2_score: 0.33407064154052113
```

In [40]:
```
# r2 first 0.11358849255891978
# r2 score second
# 0.13082727503038527
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-40-eae4de5ee47f> in <module>
      3 # 0.13082727503038527
      4
----> 5 print(confusion_matrix(y_test, preds))

NameError: name 'confusion_matrix' is not defined
```

In [41]:
```
## project

bank = pd.read_csv('bank-full.csv',sep=";")
```

In [42]:
```
bank.head(10)
```

Out[42]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	pou
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	un
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	un
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	un
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	un
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	un
5	35	management	married	tertiary	no	231	yes	no	unknown	5	may	139	1	-1	0	un
6	28	management	single	tertiary	no	447	yes	yes	unknown	5	may	217	1	-1	0	un
7	42	entrepreneur	divorced	tertiary	yes	2	yes	no	unknown	5	may	380	1	-1	0	un
8	58	retired	married	primary	no	121	yes	no	unknown	5	may	50	1	-1	0	un
9	43	technician	single	secondary	no	593	yes	no	unknown	5	may	55	1	-1	0	un

In [43]:
```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

In [44]:
```
bank.describe()
```

Out[44]:

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

In [45]:
```
bank['job'] = bank['job'].replace({'admin.':0,'blue-collar':1,'entrepreneur':2,'housemaid':3,'management':4,'re
bank['marital'] = bank['marital'].replace({'divorced':0,'married':1,'single':2,'unknown':3})
bank['default'] = bank['default'].replace({'no': 0, 'yes': 1})
bank['education'] = bank['education'].replace({"unknown":0,"secondary":1,"primary":2,"tertiary":3})
bank['housing'] = bank['housing'].replace({'yes':0,'no':1})
bank['loan'] = bank['loan'].replace({'yes':0,'no':1})
bank['contact'] = bank['contact'].replace({"unknown":0,"telephone":1,"cellular":2})
bank['month'] = bank['month'].replace({'jan':1,'feb':2,'mar':3,'apr':4,'may':5,'jun':6,'jul':7,'aug':8,'sep':9,
bank['poutcome'] = bank['poutcome'].replace({"unknown":0,"other":1,"failure":2,"success":3})
bank['y'] = bank['y'].replace({'no':0,'yes':1})
```

In [46]:
```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  int64
 2   marital    45211 non-null  int64
 3   education  45211 non-null  int64
 4   default    45211 non-null  int64
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  int64
 7   loan       45211 non-null  int64
 8   contact    45211 non-null  int64
 9   day        45211 non-null  int64
 10  month      45211 non-null  int64
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  int64
 16  y          45211 non-null  int64
dtypes: int64(17)
memory usage: 5.9 MB
```

In [47]:
```
bank.head(10)
```

Out[47]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	4	1	3	0	2143	0	1	0	5	5	261	1	-1	0	0	0
1	44	9	2	1	0	29	0	1	0	5	5	151	1	-1	0	0	0
2	33	2	1	1	0	2	0	0	0	5	5	76	1	-1	0	0	0
3	47	1	1	0	0	1506	0	1	0	5	5	92	1	-1	0	0	0
4	33	11	2	0	0	1	1	1	0	5	5	198	1	-1	0	0	0
5	35	4	1	3	0	231	0	1	0	5	5	139	1	-1	0	0	0
6	28	4	2	3	0	447	0	0	0	5	5	217	1	-1	0	0	0
7	42	2	0	3	1	2	0	1	0	5	5	380	1	-1	0	0	0
8	58	5	1	2	0	121	0	1	0	5	5	50	1	-1	0	0	0
9	43	9	2	1	0	593	0	1	0	5	5	55	1	-1	0	0	0

In [48]:
```
plt.figure(figsize=(16, 6))
_ = sns.heatmap(bank.corr(), vmin=-1, vmax=1, annot=True)
```

In [49]:
```
# trimming our data set
bank.drop(['age','job','marital','education','month','day' ],axis=1, inplace=True)
```

In [50]:
```
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 is here
X = bank[feat_cols].values
y = bank[target_col].values
```

In [51]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [54]:
```
# define and fit our model
model = KNeighborsClassifier(n_neighbors=2, n_jobs=-1)
model.fit(X_train, y_train)
```

Out[54]: `KNeighborsClassifier(n_jobs=-1, n_neighbors=2)`

In [55]:
```
# gather the predictations 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 0]
Predictions for test data set
[0 0 0 ... 0 0 0]
```

In [56]:
```
#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 0]
```

In [57]:
```
# 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 [58]:
```
# 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
```

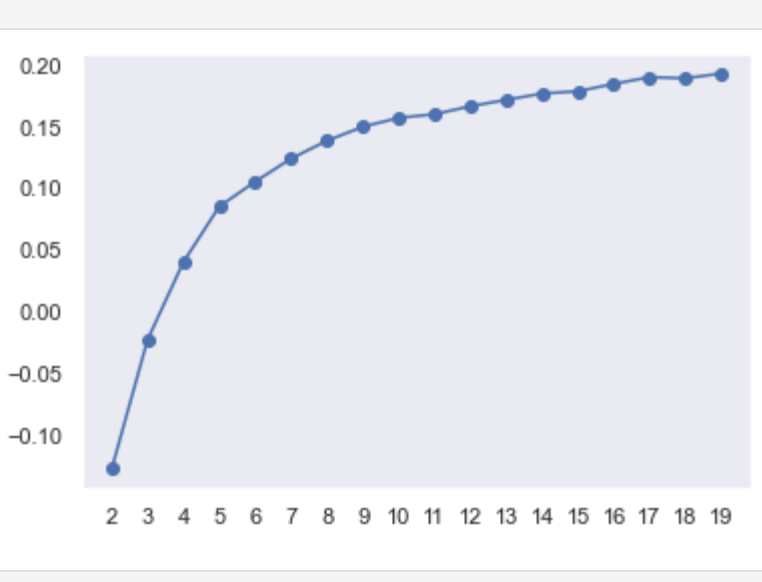
In [59]:
```
scores = []
print(f'Features: {feat_cols} \nTarget: {target_col}')

# 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', 'po
utcome']
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 [60]:
```
scores
```

Out[60]:
```
[-0.12755649963022897,
 -0.022771892071443034,
 0.04026101499003165,
 0.08523819359788398,
 0.10540756569665444,
 0.12406872867049445,
 0.13856716000632552,
 0.14989702988045117,
 0.15714956212988263,
 0.16007682816566693,
 0.16654398428200312,
 0.17177995090769793,
 0.17683477569997308,
 0.1787295742066205,
 0.1846746312689379,
 0.19018824985153637,
 0.18923267805279098,
 0.19315293150546287]
```

In [61]:
```
# display the resutls
plt.plot(range(2, 20), scores)
plt.scatter(range(2, 20), scores)
plt.grid()
_ =plt.xticks(range(2, 20))
```



In [62]:
```
# define and fit our model
model = KNeighborsRegressor(n_neighbors=17, n_jobs=-1)
model.fit(X_train, y_train)

# gather the predictations 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)
```

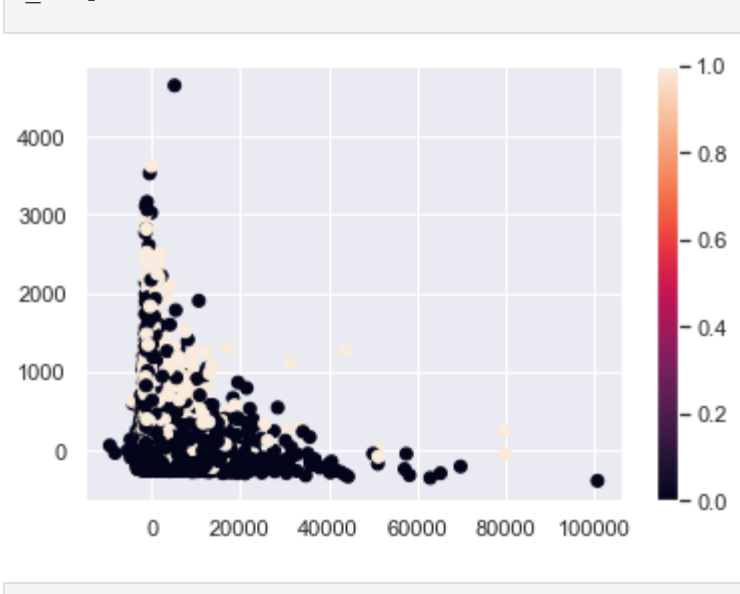
```
Actuals for test data set
[0 0 0 ... 0 0 0]
Predictions for test data set
[0.         0.05882353 0.05882353 ... 0.05882353 0.05882353 0.        ]
```

In [63]:
```
differs = y_test - preds

print(f'Differences between the two sets:\n{differs}')

print(f'r2_score: {r2_score(y_test,preds)}')
```

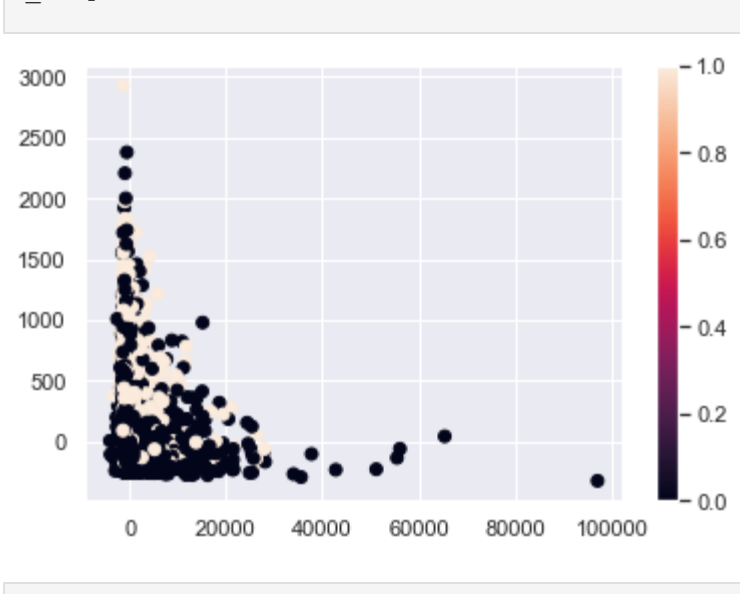
```
Differences between the two sets:
[ 0.         -0.05882353 -0.05882353 ... -0.05882353 -0.05882353
  0.        ]
r2_score: 0.19018824985153637
```

In [64]:
```
#visualiztion

pc = PCA()
tr_pca = pc.fit_transform(X_train)
te_pca = pc.transform(X_test)
```

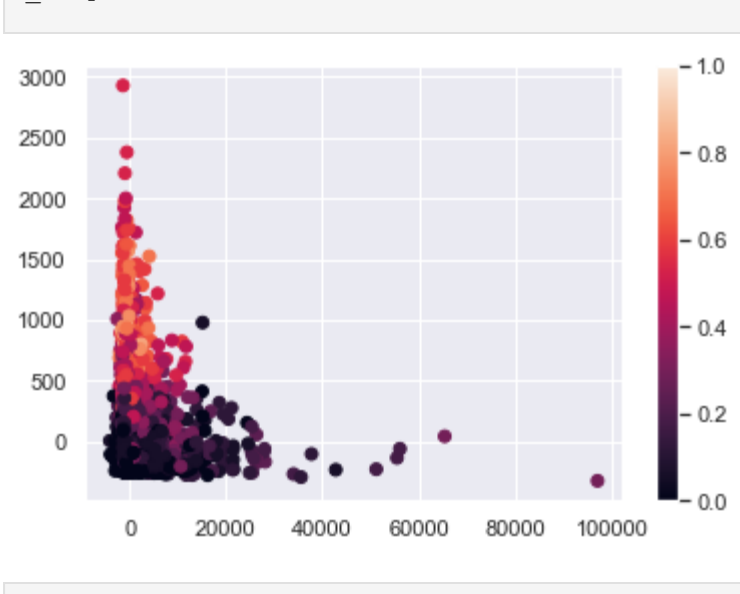
In [65]:
```
# indexing of a matrix in Python (numpy) is [rows, cols]
_ = plt.scatter(tr_pca[:, 0], tr_pca[:, 1], c=y_train)
_ = plt.colorbar()
```



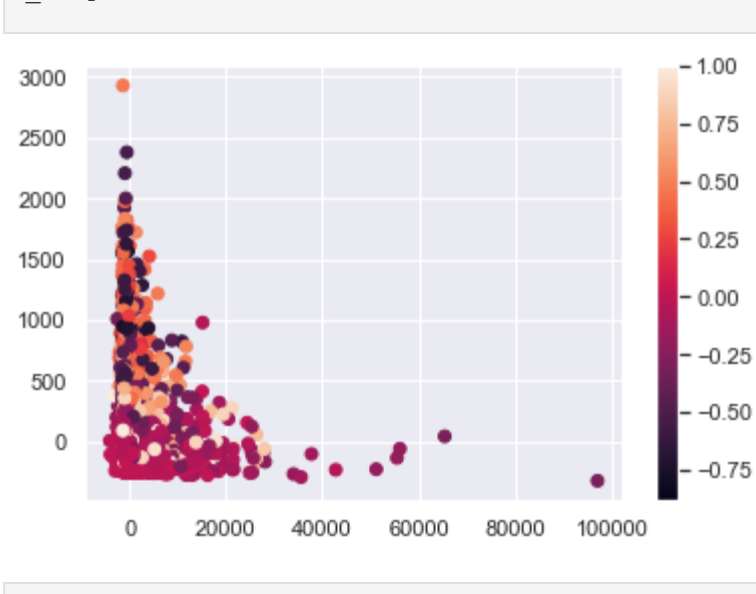
In [66]:
```
_ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=y_test)
_ = plt.colorbar()
```



In [67]:
```
_ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=preds)
_ = plt.colorbar()
```



In [68]:
```
_ = plt.scatter(te_pca[:, 0], te_pca[:, 1], c=differs)
_ = plt.colorbar()
```



In [69]:
```
# 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 [70]:
```
new_bank_tr = pd.DataFrame(X_tr_norm,columns=feat_cols)
new_bank_tr['y'] = y_train
new_bank_tr.head(10)
```

Out[70]:

	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	1.0	0.054697	0.016129	0.000000	0.000000	0.0	0
2	0.0	0.107684	1.0	1.0	1.0	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	1.0	0.052257	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 [71]:
```
new_auto_te = pd.DataFrame(X_te_norm,columns=feat_cols)
new_auto_te['y'] = y_test
new_auto_te.head(10)
```

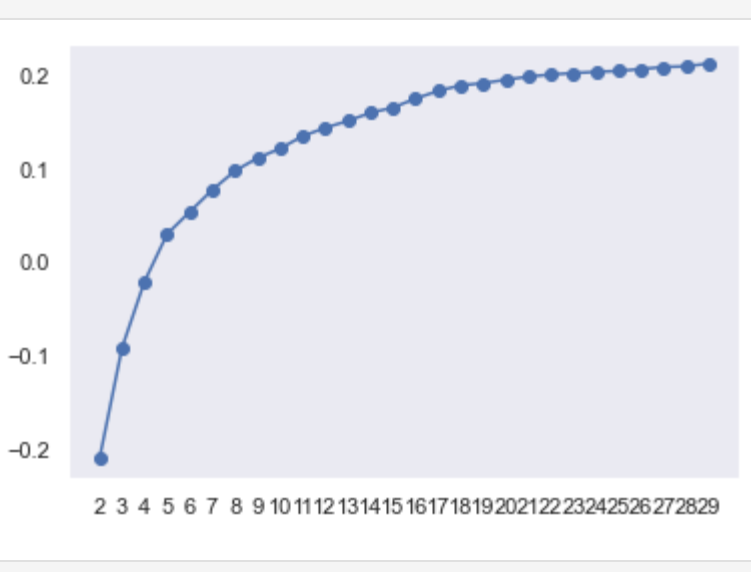
Out[71]:

	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 [74]:
```
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', 'po
utcome']
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 [76]:
```
# display the resutls
plt.plot(range(2, 30), scores_norm)
plt.scatter(range(2, 30), scores_norm)
plt.grid()
_ =plt.xticks(range(2, 30))
```



In [77]:
```
# define and fit our model with k=12
model_norm = KNeighborsRegressor(n_neighbors=21, n_jobs=-1)
model_norm.fit(X_tr_norm, y_train)

# gather the predictations 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 0]
Predictions for test data set
[0.        0.        0.0952381 ... 0.        0.        0.       ]
```

In [78]:
```
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.        -0.0952381 ...  0.         0.         0.       ]
r2_score: 0.19855738338320994
```

In [79]:
```
# confusion matrix

from sklearn.metrics import confusion_matrix
```

In [80]:
```
confusion_matrix(y_test, preds)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-80-d8a7f0b74565> in <module>
----> 1 confusion_matrix(y_test, preds)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py in confusion_matrix(y_true, y_pre
d, labels, sample_weight, normalize)
    294
    295     """
--> 296     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    297     if y_type not in ("binary", "multiclass"):
    298         raise ValueError("%s is not supported" % y_type)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py in _check_targets(y_true, y_pred)
     90
     91     if len(y_type) > 1:
---> 92         raise ValueError("Classification metrics can't handle a mix of {0} "
     93                          "and {1} targets".format(type_true, type_pred))
     94

ValueError: Classification metrics can't handle a mix of binary and continuous targets
```