ASSIGNEMENT - 2

Sir i have taken another data set as previous data set(Vaccination Data) didn't have a target variable

In [50]:
```
# import all libraries
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline

import warnings # supress warnings
warnings.filterwarnings('ignore')
```

In [23]:
```
# import Housing.csv
housing = pd.read_csv('Housing.csv')
housing.head()
```

Out[23]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	f
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	

In [24]:

```
# number of observations
len(housing.index)
```

Out[24]: 545

For the first experiment, we'll do regression with only one feature. Let's filter the data so it only contains `area` and `price`.

In [25]:

```
# filter only area and price
df = housing.loc[:, ['area', 'price']]
df.head()
```

Out[25]:

	area	price
0	7420	13300000
1	8960	12250000
2	9960	12250000
3	7500	12215000
4	7420	11410000

In [26]:

```
# recaling the variables (both)
df_columns = df.columns
scaler = MinMaxScaler()
df = scaler.fit_transform(df)

# rename columns (since now its an np array)
df = pd.DataFrame(df)
df.columns = df_columns

df.head()
```

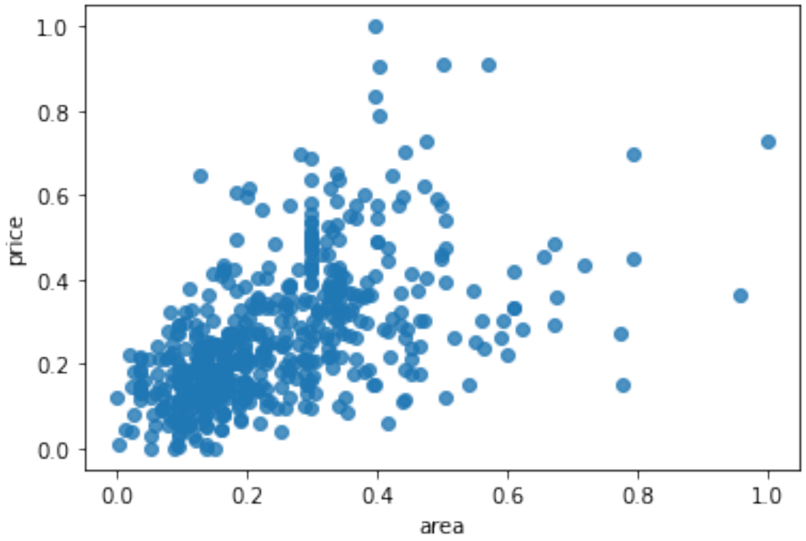
Out[26]:

	area	price
0	0.396564	1.000000
1	0.502405	0.909091
2	0.571134	0.909091
3	0.402062	0.906061
4	0.396564	0.836364

In [27]:

```
# visualise area-price relationship
sns.regplot(x="area", y="price", data=df, fit_reg=False)
```

Out[27]: <AxesSubplot:xlabel='area', ylabel='price'>



In [28]:
```
# split into train and test
df_train, df_test = train_test_split(df,
                                      train_size = 0.7,
                                      test_size = 0.3,
                                      random_state = 10)
print(len(df_train))
print(len(df_test))
```

```
381
164
```

In [29]:
```
# split into X and y for both train and test sets
# reshaping is required since sklearn requires the data to be in shape
# (n, 1), not as a series of shape (n, )
X_train = df_train['area']
X_train = X_train.values.reshape(-1, 1)
y_train = df_train['price']

X_test = df_test['area']
X_test = X_test.values.reshape(-1, 1)
y_test = df_test['price']
```

In [30]:
```
len(X_train)
```

Out[30]: 381

In [31]:
```
# data preparation

# list of all the "yes-no" binary categorical variables
# we'll map yes to 1 and no to 0
binary_vars_list =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

# defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

# applying the function to the housing variables list
housing[binary_vars_list] = housing[binary_vars_list].apply(binary_map)
housing.head()
```

Out[31]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	f
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	

In [32]:

```
# 'dummy' variables
# get dummy variables for 'furnishingstatus'
# also, drop the first column of the resulting df (since n-1 dummy vars suffice)
status = pd.get_dummies(housing['furnishingstatus'], drop_first = True)
status.head()
```

Out[32]:

	semi-furnished	unfurnished
0	0	0
1	0	0
2	1	0
3	0	0
4	0	0

In [33]:

```
# concat the dummy variable df with the main df
housing = pd.concat([housing, status], axis = 1)
housing.head()
```

Out[33]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	f
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	

In [34]:

```
# 'furnishingstatus' since we alreday have the dummy vars
housing.drop(['furnishingstatus'], axis = 1, inplace = True)
housing.head()
```

Out[34]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	f
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	

In [36]:

```
# train-test 70-30 split
df_train, df_test = train_test_split(housing,
                                     train_size = 0.7,
                                     test_size = 0.3,
                                     random_state = 100)

# rescale the features
scaler = MinMaxScaler()

# apply scaler() to all the numeric columns
numeric_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']
df_train[numeric_vars] = scaler.fit_transform(df_train[numeric_vars])
df_train.head()
```

Out[36]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	p
359	0.169697	0.155227	0.4	0.0	0.000000	1	0	0	0	0	0.333333	
19	0.615152	0.403379	0.4	0.5	0.333333	1	0	0	0	1	0.333333	
159	0.321212	0.115628	0.4	0.5	0.000000	1	1	1	0	1	0.000000	
35	0.548133	0.454417	0.4	0.5	1.000000	1	0	0	0	1	0.666667	
28	0.575758	0.538015	0.8	0.5	0.333333	1	0	1	1	0	0.666667	

Splitting Into Train and Test

In [37]:

```
# apply rescaling to the test set also
df_test[numeric_vars] = scaler.fit_transform(df_test[numeric_vars])
df_test.head()
```

Out[37]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	p
265	0.247651	0.084536	0.333333	0.000000	0.333333	1	0	0	0	0	0.000000	
54	0.530201	0.298969	0.333333	0.333333	0.333333	1	1	0	0	1	0.333333	
171	0.328859	0.592371	0.333333	0.000000	0.000000	1	0	0	0	0	0.333333	
244	0.261745	0.252234	0.333333	0.000000	0.333333	1	1	1	0	0	0.000000	
268	0.245638	0.226804	0.666667	0.000000	0.333333	1	0	0	0	1	0.000000	

In [38]:

```
# divide into X_train, y_train, X_test, y_test
y_train = df_train.pop('price')
X_train = df_train


y_test = df_test.pop('price')
X_test = df_test
```

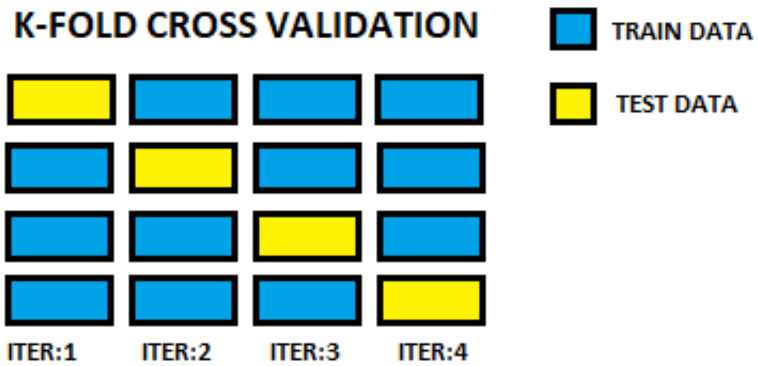
In [39]:

```
# num of max features
len(X_train.columns)
```

Out[39]: 13

Cross-Validation: A Quick Recap

The following figure illustrates k-fold cross-validation with k=4. There are some other schemes to divide the training set, we'll look at



them briefly later.

Cross-Validation in sklearn

Let's now experiment with k-fold CV.

K-Fold CV

In [40]:
```
# k-fold CV (using all the 13 variables)
lm = LinearRegression()
scores = cross_val_score(lm, X_train, y_train, scoring='r2', cv=5)
scores
```

Out[40]:
```
array([0.6829775 , 0.69324306, 0.6762109 , 0.61782891, 0.59266171])
```

In [41]:
```
# the other way of doing the same thing (more explicit)

# create a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(lm, X_train, y_train, scoring='r2', cv=folds)
scores
```

Out[41]:
```
array([0.59930574, 0.71307628, 0.61325733, 0.62739077, 0.6212937 ])
```

In [42]:
```
# number of features in X_train
len(X_train.columns)
```

Out[42]:
```
13
```

In [51]:
```
# step-1: create a cross-validation scheme
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

# step-2: specify range of hyperparameters to tune
hyper_params = [{'n_features_to_select': list(range(1, 14))}]


# step-3: perform grid search
# 3.1 specify model
lm = LinearRegression()
lm.fit(X_train, y_train)
rfe = RFE(lm)

# 3.2 call GridSearchCV()
model_cv = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 13 candidates, totalling 65 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  65 out of  65 | elapsed:    0.7s finished
```

Out[51]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
             estimator=RFE(estimator=LinearRegression()),
             param_grid=[{'n_features_to_select': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                   10, 11, 12, 13]}],
             return_train_score=True, scoring='r2', verbose=1)
```

In [52]:
```
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[52]:

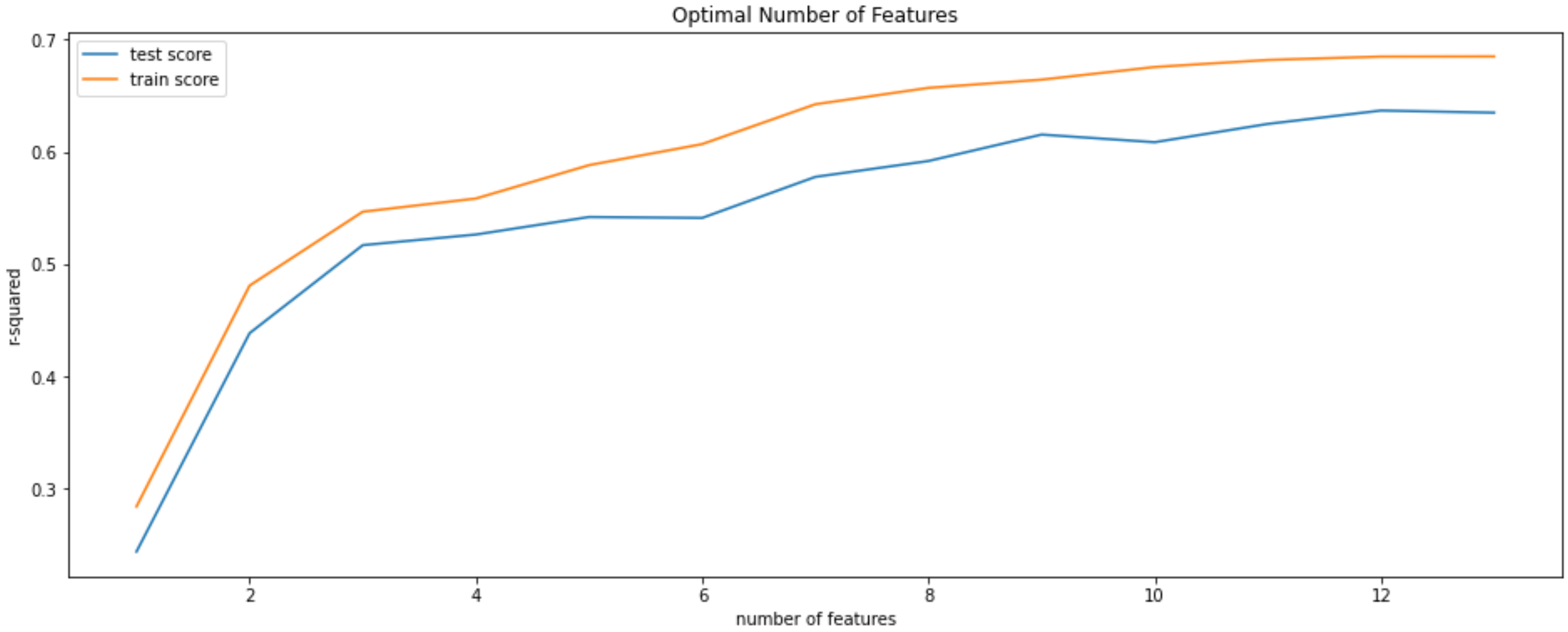
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	split0_test_score	s
0	0.012759	0.003364	0.002172	0.000239	1	{'n_features_to_select': 1}	0.172606	
1	0.013315	0.006864	0.002619	0.001147	2	{'n_features_to_select': 2}	0.335665	
2	0.008784	0.001134	0.001977	0.000360	3	{'n_features_to_select': 3}	0.421848	
3	0.007626	0.000727	0.001827	0.000261	4	{'n_features_to_select': 4}	0.449487	
4	0.006959	0.000397	0.001829	0.000106	5	{'n_features_to_select': 5}	0.494779	
5	0.006090	0.000150	0.001682	0.000090	6	{'n_features_to_select': 6}	0.512477	
6	0.005647	0.000104	0.001642	0.000063	7	{'n_features_to_select': 7}	0.568887	
7	0.005765	0.000544	0.001818	0.000263	8	{'n_features_to_select': 8}	0.570639	
8	0.005942	0.001044	0.002150	0.000632	9	{'n_features_to_select': 9}	0.578843	
9	0.004953	0.000143	0.001669	0.000066	10	{'n_features_to_select': 10}	0.574376	
10	0.004185	0.000168	0.001622	0.000105	11	{'n_features_to_select': 11}	0.578083	
11	0.003460	0.000349	0.001970	0.000310	12	{'n_features_to_select': 12}	0.602951	
12	0.002502	0.000143	0.001677	0.000142	13	{'n_features_to_select': 13}	0.599306	

13 rows × 21 columns

In [53]:
```
# plotting cv results
plt.figure(figsize=(16,6))

plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_test_score"])
plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'], loc='upper left')
```

Out[53]:
```
<matplotlib.legend.Legend at 0x7f8408194250>
```



Now we can choose the optimal value of number of features and build a final model.

In [54]:
```
# final model
n_features_optimal = 10

lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, n_features_to_select=n_features_optimal)
rfe = rfe.fit(X_train, y_train)

# predict prices of X_test
y_pred = lm.predict(X_test)
r2 = sklearn.metrics.r2_score(y_test, y_pred)
print(r2)
```

```
0.5995575338728532
```

In [ ]: