Modeling

June 22, 2021

[1]:
```
import pandas as pd
import numpy as np
import os
import pickle
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import __version__ as sklearn_version
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split, cross_validate,␣
↪GridSearchCV, learning_curve
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectKBest, f_regression
import datetime
```

[2]:
```
ski_data = pd.read_csv('ski_data_step3_features.csv')
ski_data.head()
```

[2]:
```
                  Name   Region    state  summit_elev  vertical_drop  \
0       Alyeska Resort   Alaska   Alaska         3939           2500
1  Eaglecrest Ski Area   Alaska   Alaska         2600           1540
2     Hilltop Ski Area   Alaska   Alaska         2090            294
3     Arizona Snowbowl  Arizona  Arizona        11500           2300
4  Sunrise Park Resort  Arizona  Arizona        11100           1800

   base_elev  trams  fastSixes  fastQuads  quad  …  resorts_per_100kcapita  \
0        250      1          0          2     2  …                0.410091
1       1200      0          0          0     0  …                0.410091
2       1796      0          0          0     0  …                0.410091
3       9200      0          1          0     2  …                0.027477
4       9200      0          0          1     2  …                0.027477
```

```
   resorts_per_100ksq_mile  resort_skiable_area_ac_state_ratio  \
0                 0.450867                            0.706140
1                 0.450867                            0.280702
2                 0.450867                            0.013158
3                 1.754540                            0.492708
4                 1.754540                            0.507292

   resort_days_open_state_ratio  resort_terrain_park_state_ratio  \
0                      0.434783                         0.500000
1                      0.130435                         0.250000
2                      0.434783                         0.250000
3                      0.514768                         0.666667
4                      0.485232                         0.333333

   resort_night_skiing_state_ratio  total_chairs_runs_ratio  \
0                         0.948276                 0.092105
1                              NaN                 0.111111
2                         0.051724                 0.230769
3                              NaN                 0.145455
4                         1.000000                 0.107692

   total_chairs_skiable_ratio  fastQuads_runs_ratio  fastQuads_skiable_ratio
0                    0.004348              0.026316                 0.001242
1                    0.006250              0.000000                 0.000000
2                    0.100000              0.000000                 0.000000
3                    0.010296              0.000000                 0.000000
4                    0.008750              0.015385                 0.001250

[5 rows x 36 columns]
```

[3]:
```python
ski_data.shape
```

[3]:
```
(277, 36)
```

[4]:
```python
big_mountain = ski_data[ski_data.Name == 'Big Mountain Resort']
```

[5]:
```python
big_mountain.T
```

[5]:
```
                                                 124
Name                                 Big Mountain Resort
Region                                           Montana
state                                            Montana
summit_elev                                         6817
vertical_drop                                       2353
base_elev                                           4464
trams                                                  0
```

```
fastSixes                                      0
fastQuads                                      3
quad                                           2
triple                                         6
double                                         0
surface                                        3
total_chairs                                  14
Runs                                       105.0
TerrainParks                                 4.0
LongestRun_mi                                3.3
SkiableTerrain_ac                         3000.0
Snow Making_ac                             600.0
daysOpenLastYear                           123.0
yearsOpen                                   72.0
averageSnowfall                            333.0
AdultWeekend                                81.0
projectedDaysOpen                          123.0
NightSkiing_ac                             600.0
resorts_per_state                             12
resorts_per_100kcapita                  1.122778
resorts_per_100ksq_mile                 8.161045
resort_skiable_area_ac_state_ratio      0.140121
resort_days_open_state_ratio            0.129338
resort_terrain_park_state_ratio         0.148148
resort_night_skiing_state_ratio          0.84507
total_chairs_runs_ratio                 0.133333
total_chairs_skiable_ratio              0.004667
fastQuads_runs_ratio                    0.028571
fastQuads_skiable_ratio                    0.001
```

[6]:
```
ski_data.shape
```

[6]: (277, 36)

[7]:
```
ski_data = ski_data[ski_data.Name != 'Big Mountain Resort']
```

[8]:
```
ski_data.shape
```

[8]: (276, 36)

[9]:
```
X_train, X_test, y_train, y_test = train_test_split(ski_data.
 ↪drop(columns='AdultWeekend'),
                                                    ski_data.AdultWeekend,␣
 ↪test_size=0.3,
                                                    random_state=47)
```

[10]:
```
X_train.shape, X_test.shape
```

[10]: ((193, 35), (83, 35))

[11]:
```
y_train.shape, y_test.shape
```

[11]: ((193,), (83,))

[12]:
```
names_list = ['Name', 'state', 'Region']
names_train = X_train[names_list]
names_test = X_test[names_list]
X_train.drop(columns=names_list, inplace=True)
X_test.drop(columns=names_list, inplace=True)
X_train.shape, X_test.shape
```

[12]: ((193, 32), (83, 32))

[13]:
```
X_train.dtypes
```

[13]:
```
summit_elev                            int64
vertical_drop                          int64
base_elev                              int64
trams                                  int64
fastSixes                              int64
fastQuads                              int64
quad                                   int64
triple                                 int64
double                                 int64
surface                                int64
total_chairs                           int64
Runs                                 float64
TerrainParks                         float64
LongestRun_mi                        float64
SkiableTerrain_ac                    float64
Snow Making_ac                       float64
daysOpenLastYear                     float64
yearsOpen                            float64
averageSnowfall                      float64
projectedDaysOpen                    float64
NightSkiing_ac                       float64
resorts_per_state                      int64
resorts_per_100kcapita               float64
resorts_per_100ksq_mile              float64
resort_skiable_area_ac_state_ratio   float64
resort_days_open_state_ratio         float64
resort_terrain_park_state_ratio      float64
resort_night_skiing_state_ratio      float64
total_chairs_runs_ratio              float64
total_chairs_skiable_ratio           float64
```

```
fastQuads_runs_ratio                    float64
fastQuads_skiable_ratio                 float64
dtype: object
```

[14]:
```
X_test.dtypes
```

[14]:
```
summit_elev                               int64
vertical_drop                             int64
base_elev                                 int64
trams                                     int64
fastSixes                                 int64
fastQuads                                 int64
quad                                      int64
triple                                    int64
double                                    int64
surface                                   int64
total_chairs                              int64
Runs                                    float64
TerrainParks                            float64
LongestRun_mi                           float64
SkiableTerrain_ac                       float64
Snow Making_ac                          float64
daysOpenLastYear                        float64
yearsOpen                               float64
averageSnowfall                         float64
projectedDaysOpen                       float64
NightSkiing_ac                          float64
resorts_per_state                         int64
resorts_per_100kcapita                  float64
resorts_per_100ksq_mile                 float64
resort_skiable_area_ac_state_ratio      float64
resort_days_open_state_ratio            float64
resort_terrain_park_state_ratio         float64
resort_night_skiing_state_ratio         float64
total_chairs_runs_ratio                 float64
total_chairs_skiable_ratio              float64
fastQuads_runs_ratio                    float64
fastQuads_skiable_ratio                 float64
dtype: object
```

[15]:
```
train_mean = y_train.mean()
train_mean
```

[15]:
```
63.811088082901556
```

[16]:
```
dumb_reg = DummyRegressor(strategy='mean')
dumb_reg.fit(X_train, y_train)
```

```
dumb_reg.constant_
```

[16]: array([[63.81108808]])

[17]:
```
def r_squared(y, ypred):
    """R-squared score.

    Calculate the R-squared, or coefficient of determination, of the input.

    Arguments:
    y -- the observed values
    ypred -- the predicted values
    """
    ybar = np.sum(y) / len(y) #yes, we could use np.mean(y)
    sum_sq_tot = np.sum((y - ybar)**2) #total sum of squares error
    sum_sq_res = np.sum((y - ypred)**2) #residual sum of squares error
    R2 = 1.0 - sum_sq_res / sum_sq_tot
    return R2
```

[18]:
```
y_tr_pred_ = train_mean * np.ones(len(y_train))
y_tr_pred_[:5]
```

[18]: array([63.81108808, 63.81108808, 63.81108808, 63.81108808, 63.81108808])

[19]:
```
y_tr_pred = dumb_reg.predict(X_train)
y_tr_pred[:5]
```

[19]: array([63.81108808, 63.81108808, 63.81108808, 63.81108808, 63.81108808])

[20]:
```
r_squared(y_train, y_tr_pred)
```

[20]: 0.0

[21]:
```
y_te_pred = train_mean * np.ones(len(y_test))
r_squared(y_test, y_te_pred)
```

[21]: -0.0031235200417913944

[22]:
```
def mae(y, ypred):
    """Mean absolute error.

    Calculate the mean absolute error of the arguments

    Arguments:
    y -- the observed values
    ypred -- the predicted values
    """
```

```
    abs_error = np.abs(y - ypred)
    mae = np.mean(abs_error)
    return mae
```

[23]:
```
mae(y_train, y_tr_pred)
```

[23]: 17.92346371714677

[24]:
```
mae(y_test, y_te_pred)
```

[24]: 19.136142081278486

[25]:
```
def mse(y, ypred):
    """Mean square error.

    Calculate the mean square error of the arguments

    Arguments:
    y -- the observed values
    ypred -- the predicted values
    """
    sq_error = (y - ypred)**2
    mse = np.mean(sq_error)
    return mse
```

[26]:
```
mse(y_train, y_tr_pred)
```

[26]: 614.1334096969046

[27]:
```
mse(y_test, y_te_pred)
```

[27]: 581.4365441953483

[28]:
```
np.sqrt([mse(y_train, y_tr_pred), mse(y_test, y_te_pred)])
```

[28]: array([24.78171523, 24.11299534])

[29]:
```
r2_score(y_train, y_tr_pred), r2_score(y_test, y_te_pred)
```

[29]: (0.0, -0.0031235200417913944)

[30]:
```
mean_absolute_error(y_train, y_tr_pred), mean_absolute_error(y_test, y_te_pred)
```

[30]: (17.92346371714677, 19.136142081278486)

[31]:
```
mean_squared_error(y_train, y_tr_pred), mean_squared_error(y_test, y_te_pred)
```

[31]: (614.1334096969046, 581.4365441953483)

[32]:
```
r2_score(y_train, y_tr_pred), r2_score(y_tr_pred, y_train)
```

[32]: (0.0, -3.041041349306602e+30)

[33]:
```
r2_score(y_test, y_te_pred), r2_score(y_te_pred, y_test)
```

[33]: (-0.0031235200417913944, 0.0)

[34]:
```
r_squared(y_train, y_tr_pred), r_squared(y_tr_pred, y_train)
```

[34]: (0.0, -3.041041349306602e+30)

[35]:
```
r_squared(y_test, y_te_pred), r_squared(y_te_pred, y_test)
```

```
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/ipykernel_launcher.py:13: RuntimeWarning: divide by zero encountered in
double_scalars
  del sys.path[0]
```

[35]: (-0.0031235200417913944, -inf)

[36]:
```
X_defaults_median = X_train.median()
X_defaults_median
```

[36]:
```
summit_elev                          2215.000000
vertical_drop                         750.000000
base_elev                            1300.000000
trams                                   0.000000
fastSixes                               0.000000
fastQuads                               0.000000
quad                                    1.000000
triple                                  1.000000
double                                  1.000000
surface                                 2.000000
total_chairs                            7.000000
Runs                                   28.000000
TerrainParks                            2.000000
LongestRun_mi                           1.000000
SkiableTerrain_ac                     170.000000
Snow Making_ac                         96.500000
daysOpenLastYear                      109.000000
yearsOpen                              57.000000
averageSnowfall                       120.000000
projectedDaysOpen                     115.000000
NightSkiing_ac                         70.000000
```

```
resorts_per_state                        15.000000
resorts_per_100kcapita                    0.248243
resorts_per_100ksq_mile                  22.902162
resort_skiable_area_ac_state_ratio        0.051458
resort_days_open_state_ratio              0.071225
resort_terrain_park_state_ratio           0.069444
resort_night_skiing_state_ratio           0.077081
total_chairs_runs_ratio                   0.200000
total_chairs_skiable_ratio                0.040323
fastQuads_runs_ratio                      0.000000
fastQuads_skiable_ratio                   0.000000
dtype: float64
```

[37]:
```
X_tr = X_train.fillna(X_defaults_median)
X_te = X_test.fillna(X_defaults_median)
```

[38]:
```
scaler = StandardScaler()
scaler.fit(X_tr)
X_tr_scaled = scaler.transform(X_tr)
X_te_scaled = scaler.transform(X_te)
```

[39]:
```
lm = LinearRegression().fit(X_tr_scaled, y_train)
```

[40]:
```
y_tr_pred = lm.predict(X_tr_scaled)
y_te_pred = lm.predict(X_te_scaled)
```

[41]:
```
median_r2 = r2_score(y_train, y_tr_pred), r2_score(y_test, y_te_pred)
median_r2
```

[41]:
```
(0.8177988515690603, 0.7209725843435146)
```

[42]:
```
median_mae = mean_absolute_error(y_train, y_tr_pred),␣
 ↪mean_absolute_error(y_test, y_te_pred)
median_mae
```

[42]:
```
(8.547850301825429, 9.40702011858132)
```

[43]:
```
median_mse = mean_squared_error(y_train, y_tr_pred), mean_squared_error(y_test,␣
 ↪y_te_pred)
median_mse
```

[43]:
```
(111.8958125365848, 161.73156451192267)
```

[44]:
```
X_defaults_mean = X_train.mean()
X_defaults_mean
```

[44]:
```
summit_elev                          4074.554404
vertical_drop                        1043.196891
base_elev                            3020.512953
trams                                   0.103627
fastSixes                               0.072539
fastQuads                               0.673575
quad                                    1.010363
triple                                  1.440415
double                                  1.813472
surface                                 2.497409
total_chairs                            7.611399
Runs                                   41.188482
TerrainParks                            2.434783
LongestRun_mi                           1.293122
SkiableTerrain_ac                     448.785340
Snow Making_ac                        129.601190
daysOpenLastYear                      110.100629
yearsOpen                              56.559585
averageSnowfall                       162.310160
projectedDaysOpen                     115.920245
NightSkiing_ac                         86.384615
resorts_per_state                      16.264249
resorts_per_100kcapita                  0.424802
resorts_per_100ksq_mile                40.957785
resort_skiable_area_ac_state_ratio      0.097205
resort_days_open_state_ratio            0.126014
resort_terrain_park_state_ratio         0.116022
resort_night_skiing_state_ratio         0.155024
total_chairs_runs_ratio                 0.271441
total_chairs_skiable_ratio              0.070483
fastQuads_runs_ratio                    0.010401
fastQuads_skiable_ratio                 0.001633
dtype: float64
```

[45]:
```
X_tr = X_train.fillna(X_defaults_mean)
X_te = X_test.fillna(X_defaults_mean)
```

[46]:
```
scaler = StandardScaler()
scaler.fit(X_tr)
X_tr_scaled = scaler.transform(X_tr)
X_te_scaled = scaler.transform(X_te)
```

[47]:
```
lm = LinearRegression().fit(X_tr_scaled, y_train)
```

[48]:
```
y_tr_pred = lm.predict(X_tr_scaled)
y_te_pred = lm.predict(X_te_scaled)
```

[49]:
```
r2_score(y_train, y_tr_pred), r2_score(y_test, y_te_pred)
```

[49]: (0.8170154093990025, 0.7163814716959963)

[50]:
```
mean_absolute_error(y_train, y_tr_pred), mean_absolute_error(y_test, y_te_pred)
```

[50]: (8.536884040670973, 9.416375625789271)

[51]:
```
mean_squared_error(y_train, y_tr_pred), mean_squared_error(y_test, y_te_pred)
```

[51]: (112.37695054778276, 164.39269309524346)

[52]:
```
pipe = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler(),
    LinearRegression()
)
```

[53]:
```
type(pipe)
```

[53]: sklearn.pipeline.Pipeline

[54]:
```
hasattr(pipe, 'fit'), hasattr(pipe, 'predict')
```

[54]: (True, True)

[55]:
```
pipe.fit(X_train, y_train)
```

[55]:
```
Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='median')),
                ('standardscaler', StandardScaler()),
                ('linearregression', LinearRegression())])
```

[56]:
```
y_tr_pred = pipe.predict(X_train)
y_te_pred = pipe.predict(X_test)
```

[57]:
```
r2_score(y_train, y_tr_pred), r2_score(y_test, y_te_pred)
```

[57]: (0.8177988515690603, 0.7209725843435146)

[58]:
```
median_r2
```

[58]: (0.8177988515690603, 0.7209725843435146)

[59]:
```
mean_absolute_error(y_train, y_tr_pred), mean_absolute_error(y_test, y_te_pred)
```

[59]: (8.547850301825429, 9.40702011858132)

[60]:
```
median_mae
```

[60]: (8.547850301825429, 9.40702011858132)

[61]:
```
mean_squared_error(y_train, y_tr_pred), mean_squared_error(y_test, y_te_pred)
```

[61]: (111.8958125365848, 161.73156451192267)

[62]:
```
median_mse
```

[62]: (111.8958125365848, 161.73156451192267)

[63]:
```
pipe = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler(),
    SelectKBest(f_regression),
    LinearRegression()
)
```

[64]:
```
pipe.fit(X_train, y_train)
```

[64]:
```
Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='median')),
                ('standardscaler', StandardScaler()),
                ('selectkbest',
                 SelectKBest(score_func=<function f_regression at
0x7fcd2b5a5d40>)),
                ('linearregression', LinearRegression())])
```

[65]:
```
y_tr_pred = pipe.predict(X_train)
y_te_pred = pipe.predict(X_test)
```

[66]:
```
r2_score(y_train, y_tr_pred), r2_score(y_test, y_te_pred)
```

[66]: (0.7674914326052744, 0.6259877354190837)

[67]:
```
mean_absolute_error(y_train, y_tr_pred), mean_absolute_error(y_test, y_te_pred)
```

[67]: (9.501495079727484, 11.20183019033205)

[68]:
```
pipe15 = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler(),
    SelectKBest(f_regression, k=15),
    LinearRegression()
)
```

[69]:
```
pipe15.fit(X_train, y_train)
```

[69]:
```
Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='median')),
                ('standardscaler', StandardScaler()),
                ('selectkbest',
                 SelectKBest(k=15,
                             score_func=<function f_regression at
0x7fcd2b5a5d40>)),
                ('linearregression', LinearRegression())])
```

[70]:
```
y_tr_pred = pipe15.predict(X_train)
y_te_pred = pipe15.predict(X_test)
```

[71]:
```
r2_score(y_train, y_tr_pred), r2_score(y_test, y_te_pred)
```

[71]:
```
(0.7924096060483825, 0.6376199973170795)
```

[72]:
```
mean_absolute_error(y_train, y_tr_pred), mean_absolute_error(y_test, y_te_pred)
```

[72]:
```
(9.211767769307114, 10.488246867294357)
```

[73]:
```
cv_results = cross_validate(pipe15, X_train, y_train, cv=5)
```

[74]:
```
cv_scores = cv_results['test_score']
cv_scores
```

[74]:
```
array([0.63760862, 0.72831381, 0.74443537, 0.5487915 , 0.50441472])
```

[75]:
```
np.mean(cv_scores), np.std(cv_scores)
```

[75]:
```
(0.6327128053007864, 0.09502487849877693)
```

[76]:
```
np.round((np.mean(cv_scores) - 2 * np.std(cv_scores), np.mean(cv_scores) + 2 *␣
↪np.std(cv_scores)), 2)
```

[76]:
```
array([0.44, 0.82])
```

[77]:
```
pipe.get_params().keys()
```

[77]:
```
dict_keys(['memory', 'steps', 'verbose', 'simpleimputer', 'standardscaler',
'selectkbest', 'linearregression', 'simpleimputer__add_indicator',
'simpleimputer__copy', 'simpleimputer__fill_value',
'simpleimputer__missing_values', 'simpleimputer__strategy',
'simpleimputer__verbose', 'standardscaler__copy', 'standardscaler__with_mean',
'standardscaler__with_std', 'selectkbest__k', 'selectkbest__score_func',
'linearregression__copy_X', 'linearregression__fit_intercept',
'linearregression__n_jobs', 'linearregression__normalize',
'linearregression__positive'])
```

[78]:
```
k = [k+1 for k in range(len(X_train.columns))]
grid_params = {'selectkbest__k': k}
```

[79]:
```
lr_grid_cv = GridSearchCV(pipe, param_grid=grid_params, cv=5, n_jobs=-1)
```

[80]:
```
lr_grid_cv.fit(X_train, y_train)
```

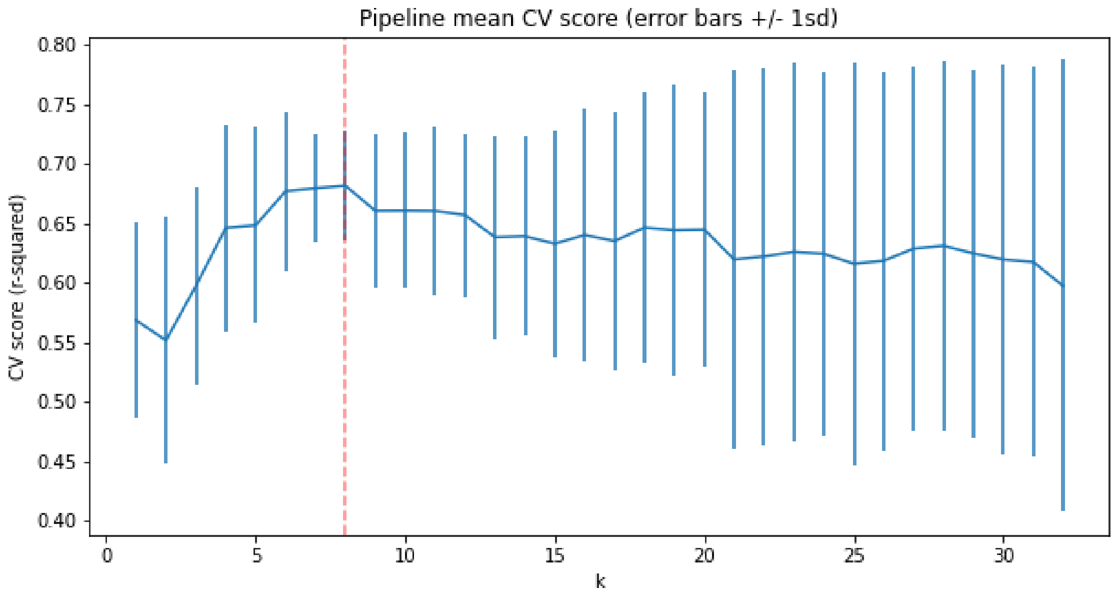
[80]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('simpleimputer',
                                        SimpleImputer(strategy='median')),
                                       ('standardscaler', StandardScaler()),
                                       ('selectkbest',
                                        SelectKBest(score_func=<function
f_regression at 0x7fcd2b5a5d40>)),
                                       ('linearregression',
                                        LinearRegression())]),
             n_jobs=-1,
             param_grid={'selectkbest__k': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                                            12, 13, 14, 15, 16, 17, 18, 19, 20,
                                            21, 22, 23, 24, 25, 26, 27, 28, 29,
                                            30, …]})
```

[81]:
```
score_mean = lr_grid_cv.cv_results_['mean_test_score']
score_std = lr_grid_cv.cv_results_['std_test_score']
cv_k = [k for k in lr_grid_cv.cv_results_['param_selectkbest__k']]
```

[82]:
```
lr_grid_cv.best_params_
```

[82]:
```
{'selectkbest__k': 8}
```

[83]:
```
best_k = lr_grid_cv.best_params_['selectkbest__k']
plt.subplots(figsize=(10, 5))
plt.errorbar(cv_k, score_mean, yerr=score_std)
plt.axvline(x=best_k, c='r', ls='--', alpha=.5)
plt.xlabel('k')
plt.ylabel('CV score (r-squared)')
plt.title('Pipeline mean CV score (error bars +/- 1sd)');
```



[84]:
```
selected = lr_grid_cv.best_estimator_.named_steps.selectkbest.get_support()
```

[85]:
```
coefs = lr_grid_cv.best_estimator_.named_steps.linearregression.coef_
features = X_train.columns[selected]
pd.Series(coefs, index=features).sort_values(ascending=False)
```

[85]:
```
vertical_drop        10.767857
Snow Making_ac        6.290074
total_chairs          5.794156
fastQuads             5.745626
Runs                  5.370555
LongestRun_mi         0.181814
trams                -4.142024
SkiableTerrain_ac    -5.249780
dtype: float64
```

[86]:
```
RF_pipe = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler(),
    RandomForestRegressor(random_state=47)
)
```

[87]:
```
rf_default_cv_results = cross_validate(RF_pipe, X_train, y_train, cv=5)
```

[88]:
```
rf_cv_scores = rf_default_cv_results['test_score']
rf_cv_scores
```

[88]: array([0.69249204, 0.78061953, 0.77546915, 0.62190924, 0.61742339])

[89]:
```
np.mean(rf_cv_scores), np.std(rf_cv_scores)
```

[89]: (0.6975826707112506, 0.07090742940774528)

[90]:
```
n_est = [int(n) for n in np.logspace(start=1, stop=3, num=20)]
grid_params = {
        'randomforestregressor__n_estimators': n_est,
        'standardscaler': [StandardScaler(), None],
        'simpleimputer__strategy': ['mean', 'median']
}
grid_params
```

[90]:
```
{'randomforestregressor__n_estimators': [10,
  12,
  16,
  20,
  26,
  33,
  42,
  54,
  69,
  88,
  112,
  143,
  183,
  233,
  297,
  379,
  483,
  615,
  784,
  1000],
 'standardscaler': [StandardScaler(), None],
 'simpleimputer__strategy': ['mean', 'median']}
```

[91]:
```
rf_grid_cv = GridSearchCV(RF_pipe, param_grid=grid_params, cv=5, n_jobs=-1)
```

[92]:
```
rf_grid_cv.fit(X_train, y_train)
```

[92]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('simpleimputer',
                                        SimpleImputer(strategy='median')),
                                       ('standardscaler', StandardScaler()),
                                       ('randomforestregressor',
RandomForestRegressor(random_state=47))]),
```

```
             n_jobs=-1,
             param_grid={'randomforestregressor__n_estimators': [10, 12, 16, 20,
                                                                  26, 33, 42, 54,
                                                                  69, 88, 112,
                                                                  143, 183, 233,
                                                                  297, 379, 483,
                                                                  615, 784,
                                                                  1000],
                         'simpleimputer__strategy': ['mean', 'median'],
                         'standardscaler': [StandardScaler(), None]})
```

[93]:
```
rf_grid_cv.best_params_
```

[93]:
```
{'randomforestregressor__n_estimators': 69,
 'simpleimputer__strategy': 'median',
 'standardscaler': None}
```

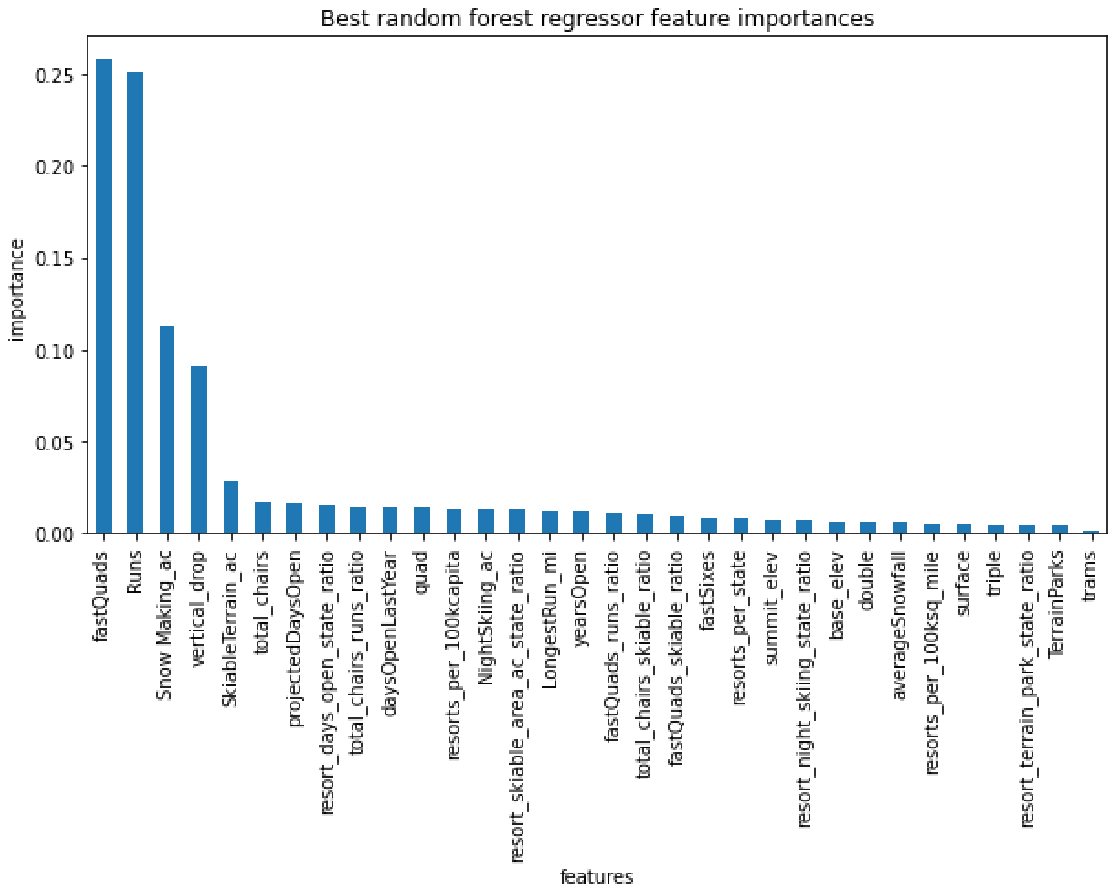
[94]:
```
rf_best_cv_results = cross_validate(rf_grid_cv.best_estimator_, X_train,␣
 ↪y_train, cv=5)
rf_best_scores = rf_best_cv_results['test_score']
rf_best_scores
```

[94]:
```
array([0.6951357 , 0.79430697, 0.77170917, 0.62254707, 0.66499334])
```

[95]:
```
np.mean(rf_best_scores), np.std(rf_best_scores)
```

[95]:
```
(0.7097384501425082, 0.06451341966873386)
```

[96]:
```
plt.subplots(figsize=(10, 5))
imps = rf_grid_cv.best_estimator_.named_steps.randomforestregressor.
 ↪feature_importances_
rf_feat_imps = pd.Series(imps, index=X_train.columns).
 ↪sort_values(ascending=False)
rf_feat_imps.plot(kind='bar')
plt.xlabel('features')
plt.ylabel('importance')
plt.title('Best random forest regressor feature importances');
```



[97]:
```
lr_neg_mae = cross_validate(lr_grid_cv.best_estimator_, X_train, y_train,
                            scoring='neg_mean_absolute_error', cv=5, n_jobs=-1)
```

[98]:
```
lr_mae_mean = np.mean(-1 * lr_neg_mae['test_score'])
lr_mae_std = np.std(-1 * lr_neg_mae['test_score'])
lr_mae_mean, lr_mae_std
```

[98]: (10.499032338015294, 1.6220608976799658)

[99]:
```
mean_absolute_error(y_test, lr_grid_cv.best_estimator_.predict(X_test))
```

[99]: 11.793465668669324

[100]:
```
rf_neg_mae = cross_validate(rf_grid_cv.best_estimator_, X_train, y_train,
                            scoring='neg_mean_absolute_error', cv=5, n_jobs=-1)
```

[101]:
```
rf_mae_mean = np.mean(-1 * rf_neg_mae['test_score'])
rf_mae_std = np.std(-1 * rf_neg_mae['test_score'])
rf_mae_mean, rf_mae_std
```

[101]: (9.644639167595688, 1.3528565172191818)

[102]:
```
mean_absolute_error(y_test, rf_grid_cv.best_estimator_.predict(X_test))
```

[102]: 9.537730050637332

[103]:
```
fractions = [.2, .25, .3, .35, .4, .45, .5, .6, .75, .8, 1.0]
train_size, train_scores, test_scores = learning_curve(pipe, X_train, y_train,␣
↪train_sizes=fractions)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
```

[104]:
```
plt.subplots(figsize=(10, 5))
plt.errorbar(train_size, test_scores_mean, yerr=test_scores_std)
plt.xlabel('Training set size')
plt.ylabel('CV scores')
plt.title('Cross-validation score as training set size increases');
```

[105]:
```
best_model = rf_grid_cv.best_estimator_
best_model.version = '1.0'
best_model.pandas_version = pd.__version__
best_model.numpy_version = np.__version__
best_model.sklearn_version = sklearn_version
best_model.X_columns = [col for col in X_train.columns]
best_model.build_datetime = datetime.datetime.now()
```

We split the data into a train set and a test set. We first define r_square, mean absolute error, and mean squared error to capture performance. To better manipulate the data, we fill the missing data with either the mean or median of the X_train, and scale all the features to zero mean and unit variance. After initial comparison, we did not find much difference in performance between using mean or median.

Later, we manipulate the technique of pipeline to fit and predict our data. To reduce the chance of overfitting, we tried different subsets of features, using one of sklearn's feature selection functions, SelectKBeat, to select k best features (default k is 10.) We also used f_regression as the score function to do the selection. After running the pipeline and evaluating the model, however, we spotted deterioration in performance. To find a better model, we tried a different k value, 15, where we found slight improvement in mean_absolute_error performance. However, if we only change the value of k, we may find our best model to be overfit. That is why we perform cross_validation to check if the model would also perform well on unseen data. We also make use of GridSearchCV to create a for loop to test out k values. And the best k value for our data is 8. We also found a list of features that are most useful:

vertical_drop 10.767857 Snow Making_ac 6.290074 total_chairs 5.794156 fastQuads 5.745626 Runs 5.370555 LongestRun_mi 0.181814 trams -4.142024 SkiableTerrain_ac -5.249780

We also fit our data in a Random Forest model and tuned some parameters of the model to improve performance. We could find similarities between the top features in the linear model and the random forest model.After comparing the performance score of the two models, we noticed that random forest has a better performance than linear model on this set of data.

[106]:
```
model = best_model
```

[107]:
```
X = ski_data.loc[ski_data.Name != "Big Mountain Resort", model.X_columns]
y = ski_data.loc[ski_data.Name != "Big Mountain Resort", 'AdultWeekend']
```

[115]:
```
len(X)
```

[115]:
```
276
```

[108]:
```
model.fit(X, y)
```

[108]:
```
Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='median')),
                ('standardscaler', None),
                ('randomforestregressor',
                 RandomForestRegressor(n_estimators=69, random_state=47))])
```

[109]:
```
cv_results = cross_validate(model, X, y, scoring='neg_mean_absolute_error',␣
 ↪cv=5, n_jobs=-1)
```

[110]:
```
cv_results['test_score']
```

[110]: array([-12.09690217, -9.30247694, -11.41595784, -8.10096706,
        -11.04942819])

[111]:
```
mae_mean, mae_std = np.mean(-1 * cv_results['test_score']), np.std(-1 *␣
↪cv_results['test_score'])
mae_mean, mae_std
```

[111]: (10.393146442687748, 1.4712769116280346)

[112]:
```
origial = pd.read_csv('ski_resort_data.csv')
```

[113]:
```
X_bm = ski_data.loc[origial.Name == "Big Mountain Resort", model.X_columns]
y_bm = ski_data.loc[origial.Name == "Big Mountain Resort", 'AdultWeekend']
```

[118]:
```
bm_pred = model.predict(X_bm).item()
print(f'Big Mountain Resort modelled price is ${bm_pred:.2f}, actual price is␣
↪${y_bm:.2f}.')
print(f'Even with the expected mean absolute error of ${mae_mean:.2f}, this␣
↪suggests .')
```

```
Big Mountain Resort modelled price is $95.87, actual price is $80.00.
Even with the expected mean absolute error of $10.39, this suggests .
```

We aim to find a more ideal ticket price for Big Mountain Resort with the model we have. The current ticket price, 'AdultWeekend', for Big Mountain is 81 dollars. We will make use of the data of other ski resorts across the country to predict the right price, assuming prices are set by a free market.

To do that, we need to refit the model with the information on Big Mountain excluded from the data. Afterwards, we use our model to make a prediction on Big Mountain, where we find out that the ticket price is underpriced. The predicted price is 95.87 dollars, which is 14.87 dollars higher than the current price. It means that there' s room for an increase in the price even with the expected mean absolute error of $10.39. Even though the model suggests an increase, we should be doubtful of this result. Our resort is heavily mispriced, by 18.36 percent. Other resorts may be the same. If a large enough number of resorts are mispriced, the model we used would be inaccurate. Moreover, we can't be sure if there's something that largely affects the ticket price but is not presented in our data. Nevertheless, we should keep a optimistic and doubtful attitude towards our result. Created in Deepnote