project_1_final_combined

April 27, 2022

0.1 Econ 412 Project 1

Alex Hong: 905857714

Gedian Wang: 705638831

Youssef Mahmoud: 905854027

Zachary DeBar: 705867064

0.1.1 Section 1: Classification

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns #visualization library
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression #problem will be solved␣
 ↪with scikit
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, LeaveOneOut, KFold,␣
 ↪cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis # LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis # QDA
from sklearn.neighbors import KNeighborsClassifier #(KNN)
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪precision_score
import statsmodels.api as sm
from IPython.core.pylabtools import figsize
import statsmodels.formula.api as smf
from patsy import dmatrices
from sklearn import datasets
%matplotlib inline
```

[2]:
```
df1 = pd.read_csv('healthcare-dataset-stroke-data.csv')
df1
```

[2]:
```
         id  gender   age  hypertension  heart_disease ever_married  \
0      9046    Male  67.0             0              1          Yes
1     51676  Female  61.0             0              0          Yes
2     31112    Male  80.0             0              1          Yes
3     60182  Female  49.0             0              0          Yes
4      1665  Female  79.0             1              0          Yes
…       …     …   …           …            …          …
5105  18234  Female  80.0             1              0          Yes
5106  44873  Female  81.0             0              0          Yes
5107  19723  Female  35.0             0              0          Yes
5108  37544    Male  51.0             0              0          Yes
5109  44679  Female  44.0             0              0          Yes

          work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0           Private          Urban             228.69  36.6  formerly smoked
1     Self-employed          Rural             202.21   NaN     never smoked
2           Private          Rural             105.92  32.5     never smoked
3           Private          Urban             171.23  34.4           smokes
4     Self-employed          Rural             174.12  24.0     never smoked
…               …            …                 …   …              …
5105        Private          Urban              83.75   NaN     never smoked
5106  Self-employed          Urban             125.20  40.0     never smoked
5107  Self-employed          Rural              82.99  30.6     never smoked
5108        Private          Rural             166.29  25.6  formerly smoked
5109       Govt_job          Urban              85.28  26.2          Unknown

      stroke
0          1
1          1
2          1
3          1
4          1
…        …
5105       0
5106       0
5107       0
5108       0
5109       0

[5110 rows x 12 columns]
```

[3]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
```

```
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

For our classification data, we chose to analyze patient data containing observations of stroke and other variables which may have been useful in predicting stroke before it occurred. This set originates from the World Health Organization. The variables are as follows:

1) id: unique identifier
2) gender: "Male", "Female" or "Other"
3) age: age of the patient
4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5) heart_disease: 0 if the patient has no heart diseases, 1 if the patient has a heart disease
6) ever_married: "No" or "Yes"
7) work_type: "children", "Govt_job", "Never_worked", "Private" or "Self-employed"
8) Residence_type: "Rural" or "Urban"
9) avg_glucose_level: average glucose level in blood
10) bmi: Body Mass Index
11) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
12) stroke: 1 if the patient suffered a stroke, 0 if patient has not suffered stroke *Note: "Unknown" in smoking_status means that the information is unavailable for this patient

[4]:
```
df1['gender'] = df1['gender'].map({'Male':0, 'Female':1})
df1['ever_married'] = df1['ever_married'].map({'No':0, 'Yes':1})
```

```
df1['work_type'] = df1['work_type'].map({'Private':0, 'Self-employed':
↪1,'Govt_job':2})
df1['Residence_type'] = df1['Residence_type'].map({'Urban':0, 'Rural':1})
df1['smoking_status'] = df1['smoking_status'].map({'never smoked':0, 'Unknown':
↪1,'formerly smoked':2,'smokes':3})
```

[5]:
```
df1.update(df1['bmi'].fillna(value=df1['bmi'].mean(), inplace=True))
```

[6]:
```
df1

df1_backup = df1
```

[7]:
```
df1.corr()
```

[7]:
```
                         id    gender       age  hypertension  heart_disease  \
id                 1.000000 -0.001929  0.003538      0.003550      -0.001296
gender            -0.001929  1.000000  0.027752     -0.021223      -0.085685
age                0.003538  0.027752  1.000000      0.276398       0.263796
hypertension       0.003550 -0.021223  0.276398      1.000000       0.108306
heart_disease     -0.001296 -0.085685  0.263796      0.108306       1.000000
ever_married       0.013690  0.030171  0.679125      0.164243       0.114644
work_type         -0.014817  0.008759  0.190557      0.047113       0.024998
Residence_type     0.001403 -0.006105 -0.014180      0.007913      -0.003092
avg_glucose_level  0.001092 -0.054722  0.238171      0.174474       0.161857
bmi                0.002999  0.025606  0.325942      0.160189       0.038899
smoking_status    -0.001713 -0.067496  0.079205      0.012531       0.063138
stroke             0.006388 -0.009081  0.245257      0.127904       0.134914

                   ever_married  work_type  Residence_type  avg_glucose_level  \
id                     0.013690  -0.014817        0.001403           0.001092
gender                 0.030171   0.008759       -0.006105          -0.054722
age                    0.679125   0.190557       -0.014180           0.238171
hypertension           0.164243   0.047113        0.007913           0.174474
heart_disease          0.114644   0.024998       -0.003092           0.161857
ever_married           1.000000   0.118094       -0.006261           0.155068
work_type              0.118094   1.000000       -0.020416           0.022348
Residence_type        -0.006261  -0.020416        1.000000           0.004946
avg_glucose_level      0.155068   0.022348        0.004946           1.000000
bmi                    0.335705   0.006788        0.000120           0.168751
smoking_status         0.085086   0.001486       -0.032112           0.025186
stroke                 0.108340   0.015050       -0.015458           0.131945

                        bmi  smoking_status    stroke
id                 0.002999       -0.001713  0.006388
gender             0.025606       -0.067496 -0.009081
age                0.325942        0.079205  0.245257
hypertension       0.160189        0.012531  0.127904
```

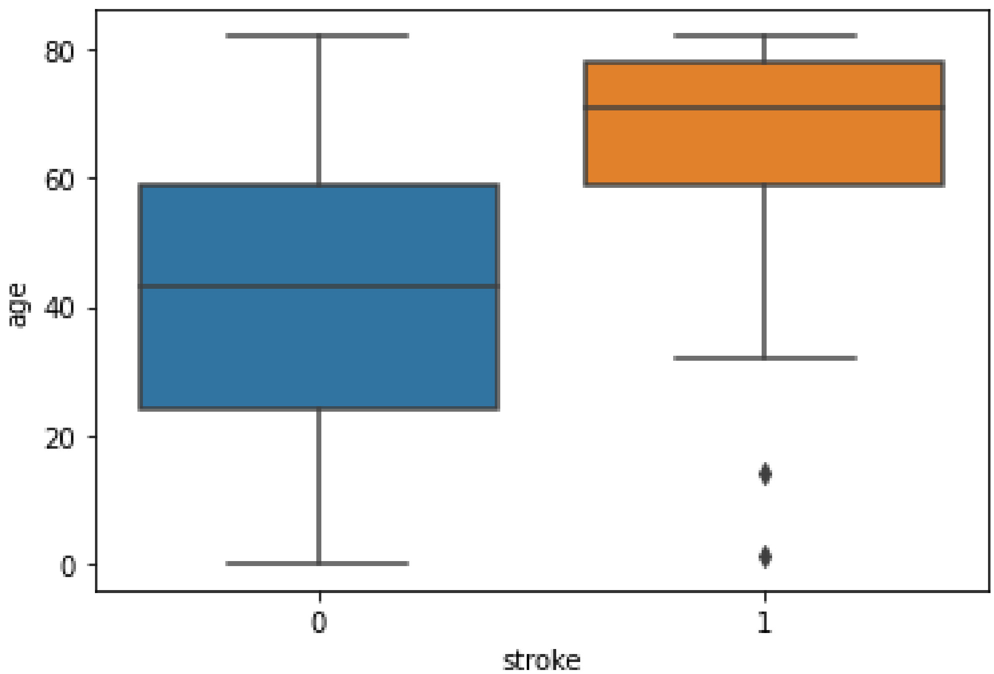
```
heart_disease       0.038899          0.063138  0.134914
ever_married        0.335705          0.085086  0.108340
work_type           0.006788          0.001486  0.015050
Residence_type      0.000120         -0.032112 -0.015458
avg_glucose_level   0.168751          0.025186  0.131945
bmi                 1.000000          0.046660  0.038947
smoking_status      0.046660          1.000000  0.030682
stroke              0.038947          0.030682  1.000000
```

Based on the correlation plot, the variables age, hypertension, heart disease and average glucose level show the highest correlation with our dependent variable, stroke.

[8]:
```
sns.pairplot(df1);
```

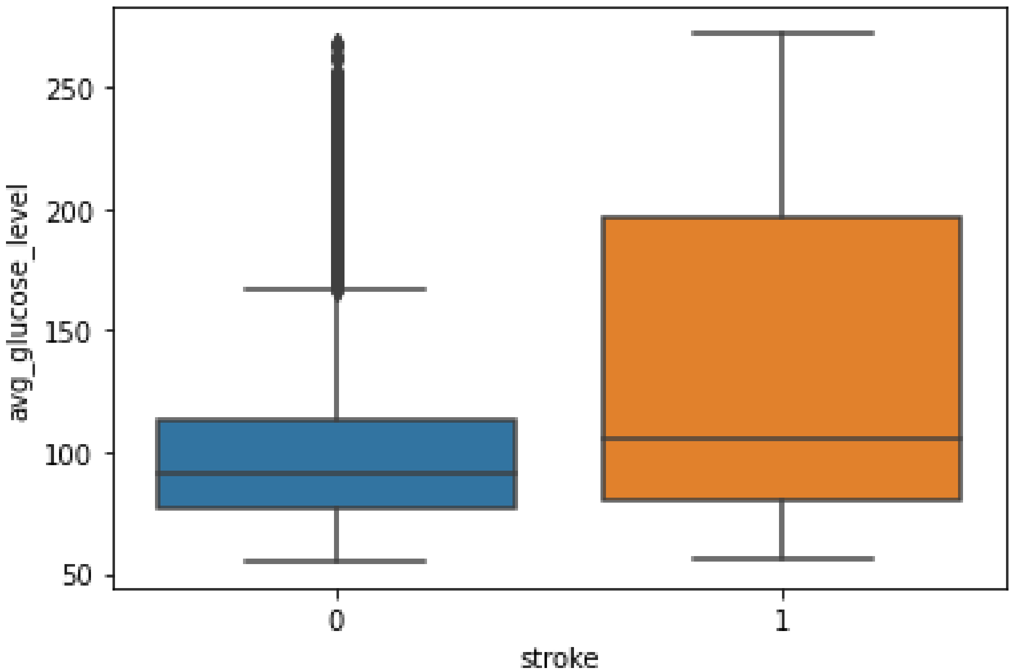
Based on the pairplots, the variable age implies that for ages below 40, the chance of getting a stroke is very low, but varies for ages above 40. The average glucose level shows alot of variation so it is difficult to predict a pattern that shows the direction of the relationship with getting a stroke. The pairplots for hypertension and heart_disease do not show any obvious implications. The BMI variable implies that when its value is under 20, its unlikely to get a stroke.

[9]:
```
sns.boxplot(x='stroke', y='age', data=df1);
```



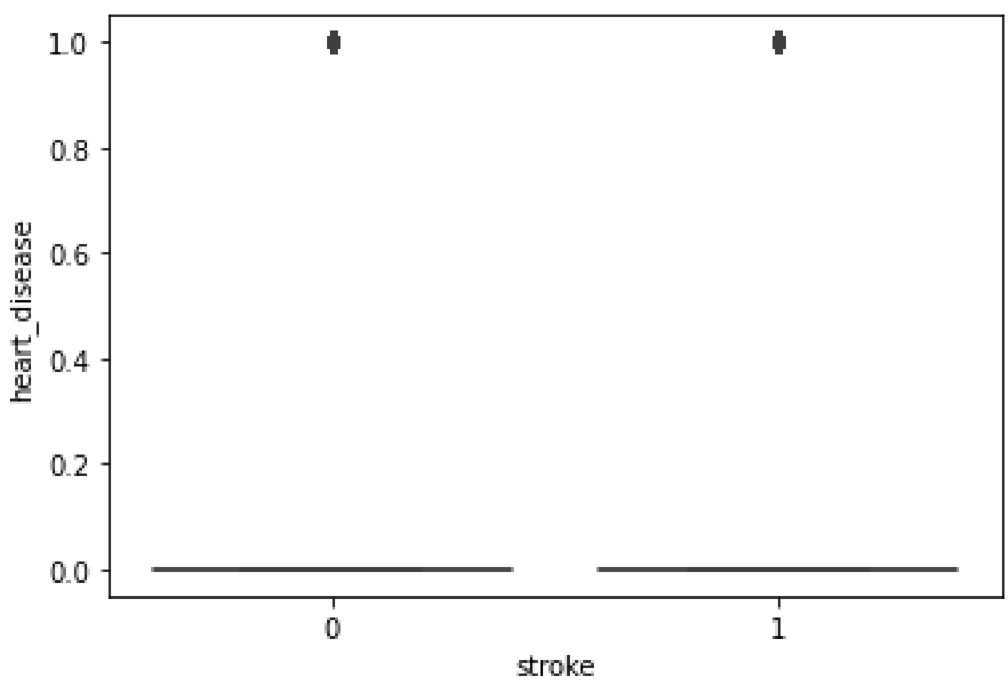
The boxplots support our previous conclusion that strokes are more likely to occur at higher ages.

[10]:
```
sns.boxplot(x='stroke', y='avg_glucose_level', data=df1);
```

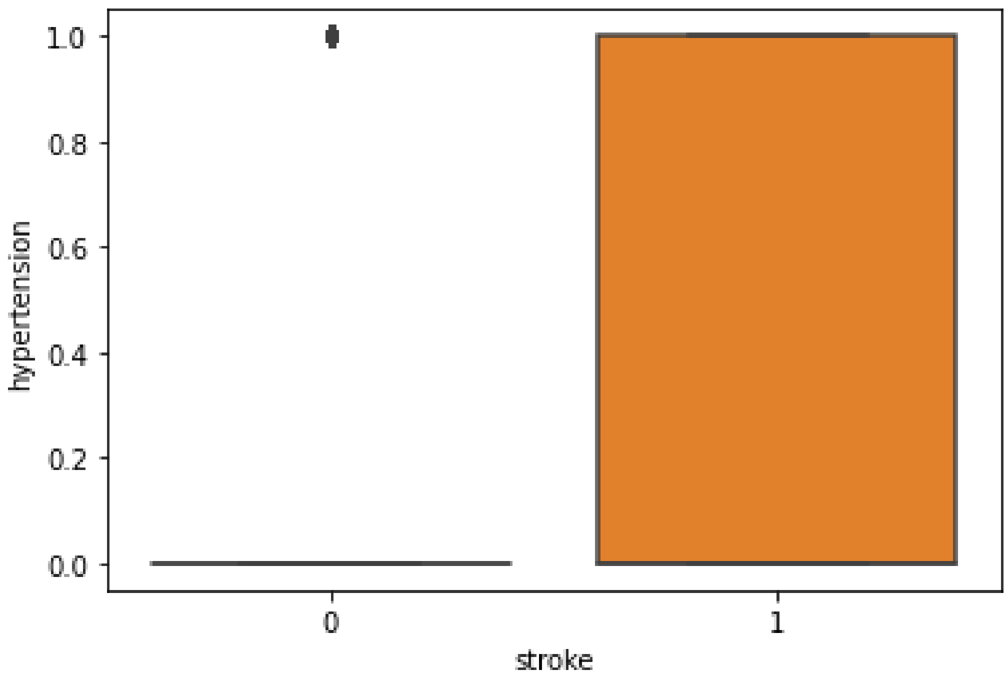


The boxplot implies that it is more likely for a stroke to occur as the average level of glucose rises.

[11]:
```
sns.boxplot(x='stroke', y='heart_disease', data=df1);
```



[12]:
```
sns.boxplot(x='stroke', y='hypertension', data=df1);
```



The boxplots for hypertension and heart diease are not informative because they are both binary variables, unlike age and average glucose level.

Now a logistic regression model will be run to finalize the choice of the best predictors

[13]:
```
y, X = dmatrices('stroke ~ bmi + avg_glucose_level + age + smoking_status +␣
↪gender + ever_married + work_type + Residence_type + hypertension +␣
↪heart_disease', data=df1, return_type='dataframe')

logit = sm.Logit(y, X)
results_logit = logit.fit()
print(results_logit.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.176862
         Iterations 9
                           Logit Regression Results
==============================================================================
Dep. Variable:                 stroke   No. Observations:                 4400
Model:                          Logit   Df Residuals:                     4389
Method:                           MLE   Df Model:                           10
Date:                Wed, 27 Apr 2022   Pseudo R-squ.:                  0.1820
```

```
Time:                        01:22:54   Log-Likelihood:                -778.19
converged:                       True   LL-Null:                       -951.29
Covariance Type:            nonrobust   LLR p-value:                 2.570e-68
================================================================================
=====
                        coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-----
Intercept            -7.5923      0.579    -13.115      0.000      -8.727
-6.458
bmi                   0.0013      0.011      0.112      0.910      -0.021
0.024
avg_glucose_level     0.0042      0.001      3.475      0.001       0.002
0.007
age                   0.0725      0.006     12.884      0.000       0.062
0.084
smoking_status        0.1017      0.062      1.629      0.103      -0.021
0.224
gender               -0.0359      0.142     -0.253      0.801      -0.314
0.243
ever_married         -0.1764      0.225     -0.783      0.433      -0.618
0.265
work_type            -0.1457      0.100     -1.464      0.143      -0.341
0.049
Residence_type       -0.0887      0.139     -0.639      0.523      -0.361
0.183
hypertension          0.3880      0.164      2.370      0.018       0.067
0.709
heart_disease         0.2749      0.191      1.441      0.150      -0.099
0.649
================================================================================
=====
```

After performing initial logit regression across all variables, we found "age", "avg_glucose_level", and "hypertension" to be significant predictors of "stroke" outcome. We continued to develop our predictive models with these variables.

[14]:
```
#Partitioning the dataset
df1_50 = df1[(df1['age']<=50)]
df1_82 = df1[(df1['age'] >50) & (df1['age'] <=82)]
```

[15]:
```
lr = LogisticRegression()
# Training set
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
# Logistic Fit
mod = lr.fit(X,df1_50['stroke'])
```

[16]:
```python
# Testing Set
X_test = df1_82[['avg_glucose_level', 'age','hypertension']]

# Confusion matrix
conf_mat = confusion_matrix(df1_82['stroke'], lr.predict(X_test))
print(conf_mat)
#overall fraction of correct predictions
lr.score(X_test, df1_82['stroke'])
print('Accuracy =', lr.score(X_test, df1_82['stroke']))
```

```
[[1897    4]
 [ 226    0]]
Accuracy = 0.8918664786083685
```

[17]:
```python
#LDA
# Training set
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]

# LDA Fit
lda = LinearDiscriminantAnalysis()
lda.fit(X,df1_50['stroke'])
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
              solver='svd', store_covariance=False, tol=0.0001)
```

[17]:
```
LinearDiscriminantAnalysis()
```

[18]:
```python
# Testing Set
X_test = df1_82[['avg_glucose_level', 'age','hypertension']]

# Confusion matrix
conf_mat = confusion_matrix(df1_82['stroke'], lda.predict(X_test))
print(conf_mat)
lda.score(X_test, df1_82['stroke'])
print('Accuracy =', lda.score(X_test, df1_82['stroke']))
```

```
[[1860   41]
 [ 212   14]]
Accuracy = 0.8810531264692054
```

[19]:
```python
#QDA
# Training set
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]

# QDA Fit
qda = QuadraticDiscriminantAnalysis()
qda.fit(X,df1_50['stroke'])
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
               store_covariance=False, tol=0.0001)
```

[19]:
```
QuadraticDiscriminantAnalysis()
```

[20]:
```python
# Testing Set
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

# Confusion matrix
conf_mat = confusion_matrix(df1_82['stroke'], qda.predict(X_test))
print(conf_mat)
qda.score(X_test, df1_82['stroke'])
print('Accuracy =', qda.score(X_test, df1_82['stroke']))
```

```
[[1529  372]
 [ 157   69]]
Accuracy = 0.7512929007992478
```

[21]:
```python
#KNN, n=1
# Training set
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]

# KNN Fit
nbrs = KNeighborsClassifier(n_neighbors=1)
nbrs.fit(X,df1_50['stroke'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

[21]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=1)
```

[22]:
```python
# Testing Set
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

# Confusion matrix
conf_mat = confusion_matrix(df1_82['stroke'], nbrs.predict(X_test))
print(conf_mat)
nbrs.score(X_test, df1_82['stroke'])
print('Accuracy =', nbrs.score(X_test, df1_82['stroke']))
```

```
[[1824   77]
 [ 214   12]]
Accuracy = 0.8631875881523272
```

[23]:
```python
#KNN,n=2
# Training set
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]

# KNN Fit
nbrs2 = KNeighborsClassifier(n_neighbors=2)
nbrs2.fit(X,df1_50['stroke'])
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

[23]: KNeighborsClassifier(n_jobs=1, n_neighbors=2)

[24]:
```
# Testing Set
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

# Confusion matrix
conf_mat = confusion_matrix(df1_82['stroke'], nbrs2.predict(X_test))
print(conf_mat)
nbrs2.score(X_test, df1_82['stroke'])
print('Accuracy =', nbrs2.score(X_test, df1_82['stroke']))
```

```
[[1892    9]
 [ 225    1]]
Accuracy = 0.8899858956276445
```

[25]:
```
#KNN,n=3
# Training set
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]

# KNN Fit
nbrs3 = KNeighborsClassifier(n_neighbors=3)
nbrs3.fit(X,df1_50['stroke'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform')
```

[25]: KNeighborsClassifier(n_jobs=1, n_neighbors=3)

[26]:
```
# Testing Set
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

# Confusion matrix
conf_mat = confusion_matrix(df1_82['stroke'], nbrs3.predict(X_test))
print(conf_mat)
nbrs3.score(X_test, df1_82['stroke'])
print('Accuracy =', nbrs3.score(X_test, df1_82['stroke']))
```

```
[[1886   15]
 [ 222    4]]
Accuracy = 0.8885754583921015
```

[27]:
```
#KNN,n=4
# Training set
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
```

```
# KNN Fit
nbrs4 = KNeighborsClassifier(n_neighbors=4)
nbrs4.fit(X,df1_50['stroke'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='uniform')
```

[27]: KNeighborsClassifier(n_jobs=1, n_neighbors=4)

[28]:
```
# Testing Set
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

# Confusion matrix
conf_mat = confusion_matrix(df1_82['stroke'], nbrs4.predict(X_test))
print(conf_mat)
nbrs4.score(X_test, df1_82['stroke'])
print('Accuracy =', nbrs4.score(X_test, df1_82['stroke']))
```

```
[[1901    0]
 [ 226    0]]
Accuracy = 0.8937470615890927
```

[29]:
```
#KNN,n=5
# Training set
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]

# KNN Fit
nbrs5 = KNeighborsClassifier(n_neighbors=5)
nbrs5.fit(X,df1_50['stroke'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

[29]: KNeighborsClassifier(n_jobs=1)

[30]:
```
# Testing Set
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

# Confusion matrix
conf_mat = confusion_matrix(df1_82['stroke'], nbrs5.predict(X_test))
print(conf_mat)
nbrs.score(X_test, df1_82['stroke'])
print('Accuracy =', nbrs5.score(X_test, df1_82['stroke']))
```

```
[[1901    0]
 [ 226    0]]
Accuracy = 0.8937470615890927
```

According to the accuracies of the models ran, KNN4 and KNN5 have the highest accuracy amongst all models. Both models had the same accuracy and confusion matrix, so we will use Bootstrap to find the model with lowest MSE.

0.1.2 Bootstrap Performance Evaluation

[31]:
```
models_mse = []
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

for i in range(100):

    boot = sklearn.utils.resample(X, replace = False, n_samples = 2127,␣
 ↪random_state = i)
    boot_Y = sklearn.utils.resample(df1_50["stroke"], replace = False,␣
 ↪n_samples = 2127, random_state = i)
    lr.fit(boot, boot_Y)
    MSE = mean_squared_error(df1_82['stroke'], lr.predict(X_test))
    models_mse.append(MSE)
print("Bootstrapped MSE={}".format(sum(models_mse)/100))
```

```
Bootstrapped MSE=0.1310296191819465
```

The bootstrapped MSE for logistic regression is 0.13103.

[32]:
```
models_mse = []
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

for i in range(100):

    boot = sklearn.utils.resample(X, replace = False, n_samples = 2127,␣
 ↪random_state = i)
    boot_Y = sklearn.utils.resample(df1_50["stroke"], replace = False,␣
 ↪n_samples = 2127, random_state = i)
    qda.fit(boot, boot_Y)
    MSE = mean_squared_error(df1_82['stroke'], qda.predict(X_test))
    models_mse.append(MSE)
print("Bootstrapped MSE={}".format(sum(models_mse)/100))
```

```
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: divide by zero
encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
```

```
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: invalid value
encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:836: RuntimeWarning: divide by zero
encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: divide by zero
encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: invalid value
encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:836: RuntimeWarning: divide by zero
encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: divide by zero
encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: invalid value
encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:836: RuntimeWarning: divide by zero
encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: divide by zero
encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
```

```
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: invalid value
encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:836: RuntimeWarning: divide by zero
encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
```

```
Bootstrapped MSE=0.28128819934179583
```

```
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: divide by zero
encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: invalid value
encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:836: RuntimeWarning: divide by zero
encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: divide by zero
encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:833: RuntimeWarning: invalid value
encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\discriminant_analysis.py:836: RuntimeWarning: divide by zero
encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
```

The bootstrapped MSE for QDA is 0.28129.

[33]:
```
models_mse = []
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]
```

```
for i in range(100):

    boot = sklearn.utils.resample(X, replace = False, n_samples = 2127,␣
 ↪random_state = i)
    boot_Y = sklearn.utils.resample(df1_50["stroke"], replace = False,␣
 ↪n_samples = 2127, random_state = i)
    lda.fit(boot, boot_Y)
    MSE = mean_squared_error(df1_82['stroke'], lda.predict(X_test))
    models_mse.append(MSE)
print("Bootstrapped MSE={}".format(sum(models_mse)/100))
```

```
Bootstrapped MSE=0.1445463093559004
```

The bootstrapped MSE for LDA is 0.14455.

[34]:
```
models_mse = []
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

for i in range(100):

    boot = sklearn.utils.resample(X, replace = False, n_samples = 2127,␣
 ↪random_state = i)
    boot_Y = sklearn.utils.resample(df1_50["stroke"], replace = False,␣
 ↪n_samples = 2127, random_state = i)
    nbrs.fit(boot, boot_Y)
    MSE = mean_squared_error(df1_82['stroke'], nbrs.predict(X_test))
    models_mse.append(MSE)
print("Bootstrapped MSE={}".format(sum(models_mse)/100))
```

```
Bootstrapped MSE=0.1335731076633757
```

The bootstrapped MSE for KNN-1 is 0.13357.

[35]:
```
models_mse = []
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

for i in range(100):

    boot = sklearn.utils.resample(X, replace = False, n_samples = 2127,␣
 ↪random_state = i)
    boot_Y = sklearn.utils.resample(df1_50["stroke"], replace = False,␣
 ↪n_samples = 2127, random_state = i)
    nbrs2.fit(boot, boot_Y)
    MSE = mean_squared_error(df1_82['stroke'], nbrs2.predict(X_test))
    models_mse.append(MSE)
print("Bootstrapped MSE={}".format(sum(models_mse)/100))
```

```
Bootstrapped MSE=0.10827456511518584
```

The bootstrapped MSE for KNN-2 is 0.10827.

[36]:
```python
models_mse = []
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

for i in range(100):

    boot = sklearn.utils.resample(X, replace = False, n_samples = 2127,␣
 ↪random_state = i)
    boot_Y = sklearn.utils.resample(df1_50["stroke"], replace = False,␣
 ↪n_samples = 2127, random_state = i)
    nbrs3.fit(boot, boot_Y)
    MSE = mean_squared_error(df1_82['stroke'], nbrs3.predict(X_test))
    models_mse.append(MSE)
print("Bootstrapped MSE={}".format(sum(models_mse)/100))
```

```
Bootstrapped MSE=0.10914433474377055
```

The bootstrapped MSE for KNN-3 is 0.109144.

[37]:
```python
models_mse = []
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

for i in range(100):

    boot = sklearn.utils.resample(X, replace = False, n_samples = 2127,␣
 ↪random_state = i)
    boot_Y = sklearn.utils.resample(df1_50["stroke"], replace = False,␣
 ↪n_samples = 2127, random_state = i)
    nbrs4.fit(boot, boot_Y)
    MSE = mean_squared_error(df1_82['stroke'], nbrs4.predict(X_test))
    models_mse.append(MSE)
print("Bootstrapped MSE={}".format(sum(models_mse)/100))
```

```
Bootstrapped MSE=0.10625293841090754
```

The bootstrapped MSE for KNN-4 is 0.10625.

[38]:
```python
models_mse = []
X = df1_50[['avg_glucose_level', 'age', 'hypertension']]
X_test = df1_82[['avg_glucose_level', 'age', 'hypertension']]

for i in range(100):

    boot = sklearn.utils.resample(X, replace = False, n_samples = 2127,␣
 ↪random_state = i)
```

```
    boot_Y = sklearn.utils.resample(df1_50["stroke"], replace = False,␣
↪n_samples = 2127, random_state = i)
    nbrs5.fit(boot, boot_Y)
    MSE = mean_squared_error(df1_82['stroke'], nbrs5.predict(X_test))
    models_mse.append(MSE)
print("Bootstrapped MSE={}".format(sum(models_mse)/100))
```

```
Bootstrapped MSE=0.10625293841090754
```

The bootstrapped MSE for KNN-5 is 0.10625.

Based on the results of our bootstrap analysis, KNN-4 and KNN-5 are the models which produce the minimized bootstrap Mean Square Error, both tied at 0.10625.

0.2 Section 2: Regularization

[39]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV, Lasso,␣
↪LassoCV, ElasticNet, ElasticNetCV
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
%matplotlib inline
plt.style.use('seaborn-white')
```

[40]:
```
data = pd.read_csv('Movie_classification.csv')
data.index.name = 'movie'
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Marketing expense    506 non-null    float64
 1   Production expense   506 non-null    float64
 2   Multiplex coverage   506 non-null    float64
 3   Budget               506 non-null    float64
 4   Movie_length         506 non-null    float64
 5   Lead_ Actor_Rating   506 non-null    float64
 6   Lead_Actress_rating  506 non-null    float64
```

```
 7   Director_rating      506 non-null    float64
 8   Producer_rating      506 non-null    float64
 9   Critic_rating        506 non-null    float64
 10  Trailer_views        506 non-null    int64
 11  3D_available         506 non-null    object
 12  Time_taken           494 non-null    float64
 13  Twitter_hastags      506 non-null    float64
 14  Genre                506 non-null    object
 15  Avg_age_actors       506 non-null    int64
 16  Num_multiplex        506 non-null    int64
 17  Collection           506 non-null    int64
 18  Start_Tech_Oscar     506 non-null    int64
dtypes: float64(12), int64(5), object(2)
memory usage: 75.2+ KB
```

Our objetive in analyzing this data is to identify if there is a functional prediction model to predict whether or not a film will qualify for an Oscar award, as indicated with the "Start_Tech_Oscar" variable.

Data source:

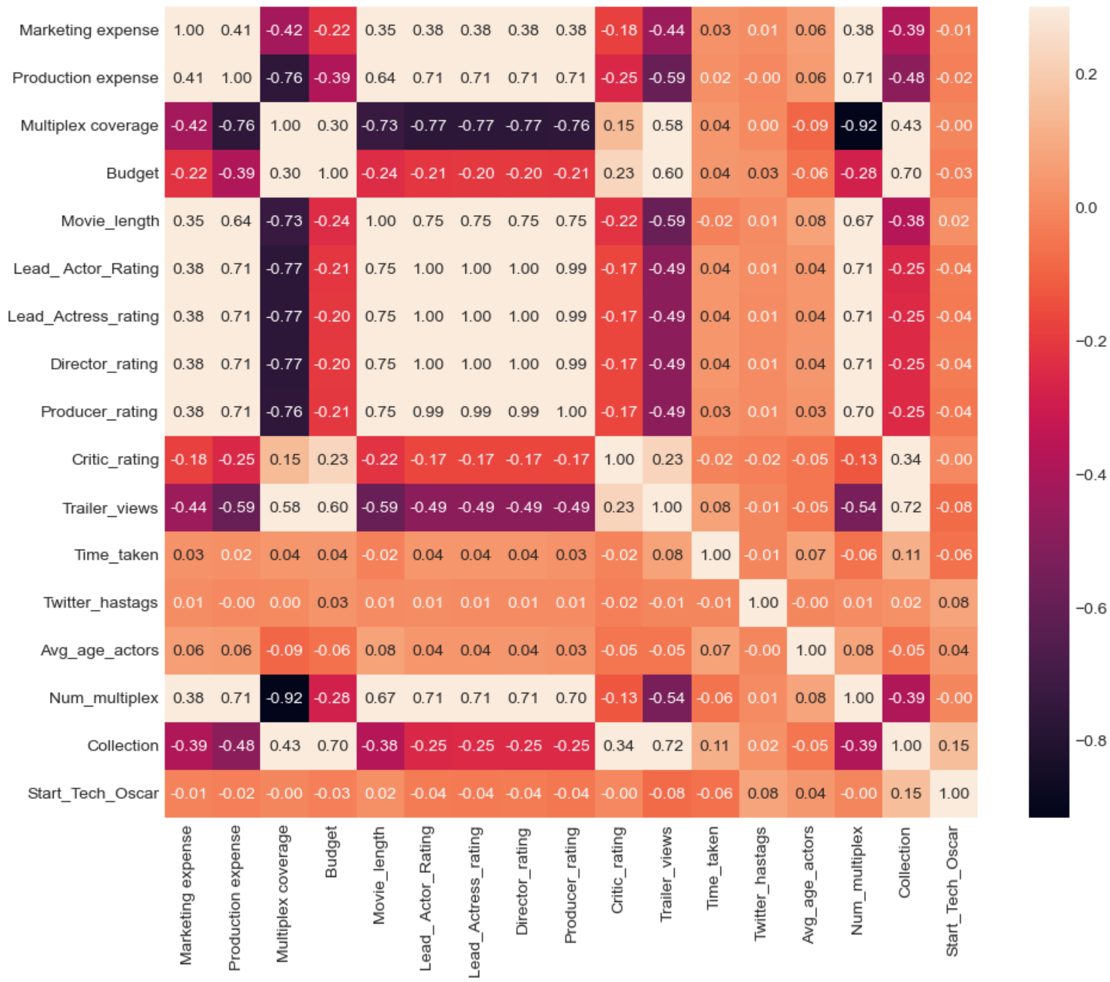
https://www.kaggle.com/datasets/balakrishcodes/others?select=Movie_classification.csv

[41]:
```
data
```

[41]:
```
       Marketing expense  Production expense  Multiplex coverage     Budget  \
movie
0                20.1264               59.62               0.462  36524.125
1                20.5462               69.14               0.531  35668.655
2                20.5458               69.14               0.531  39912.675
3                20.6474               59.36               0.542  38873.890
4                21.3810               59.36               0.542  39701.585
…                  …                   …                   …          …
501              21.2526               78.86               0.427  36624.115
502              20.9054               78.86               0.427  33996.600
503              21.2152               78.86               0.427  38751.680
504              22.1918               78.86               0.427  37740.670
505              20.9482               78.86               0.427  33496.650

       Movie_length  Lead_ Actor_Rating  Lead_Actress_rating  Director_rating  \
movie
0             138.7               7.825                8.095            7.910
1             152.4               7.505                7.650            7.440
2             134.6               7.485                7.570            7.495
3             119.3               6.895                7.035            6.920
4             127.7               6.920                7.070            6.815
…               …                   …                    …                …
501           142.6               8.680                8.775            8.620
502           150.2               8.780                8.945            8.770
```

```
503          164.5               8.830               8.970            8.855
504          162.8               8.730               8.845            8.800
505          154.3               8.640               8.880            8.680

       Producer_rating  Critic_rating  Trailer_views 3D_available  Time_taken  \
movie
0                7.995           7.94         527367          YES      109.60
1                7.470           7.44         494055           NO      146.64
2                7.515           7.44         547051           NO      147.88
3                7.020           8.26         516279          YES      185.36
4                7.070           8.26         531448           NO      176.48
…                  …             …             …          …            …
501              8.970           6.80         492480           NO      186.96
502              8.930           7.80         482875          YES      132.24
503              9.010           7.80         532239           NO      109.56
504              8.845           6.80         496077          YES      158.80
505              8.790           6.80         518438          YES      205.60

       Twitter_hastags     Genre  Avg_age_actors  Num_multiplex  Collection  \
movie
0              223.840  Thriller              23            494       48000
1              243.456     Drama              42            462       43200
2             2022.400    Comedy              38            458       69400
3              225.344     Drama              45            472       66800
4              225.792     Drama              55            395       72400
…                  …       …               …             …          …
501            243.584    Action              27            561       44800
502            263.296    Action              20            600       41200
503            243.824    Comedy              31            576       47800
504            303.520    Comedy              47            607       44000
505            203.040    Comedy              45            604       38000

       Start_Tech_Oscar
movie
0                     1
1                     0
2                     1
3                     1
4                     1
…                  …
501                   0
502                   0
503                   0
504                   0
505                   0

[506 rows x 19 columns]
```

[42]:
```python
# Correlation plot

correlations = data.corr()              # 17*17 matrix of corr coefficients
plt.figure(figsize=(11, 9),dpi=100)     # picture size
sns.heatmap(data = correlations,
           vmax=0.3,                    # color depth
           annot=True,                  # add number
           fmt=".2f",                   # round digit
           annot_kws={'size':10})       # word size
plt.show()
```

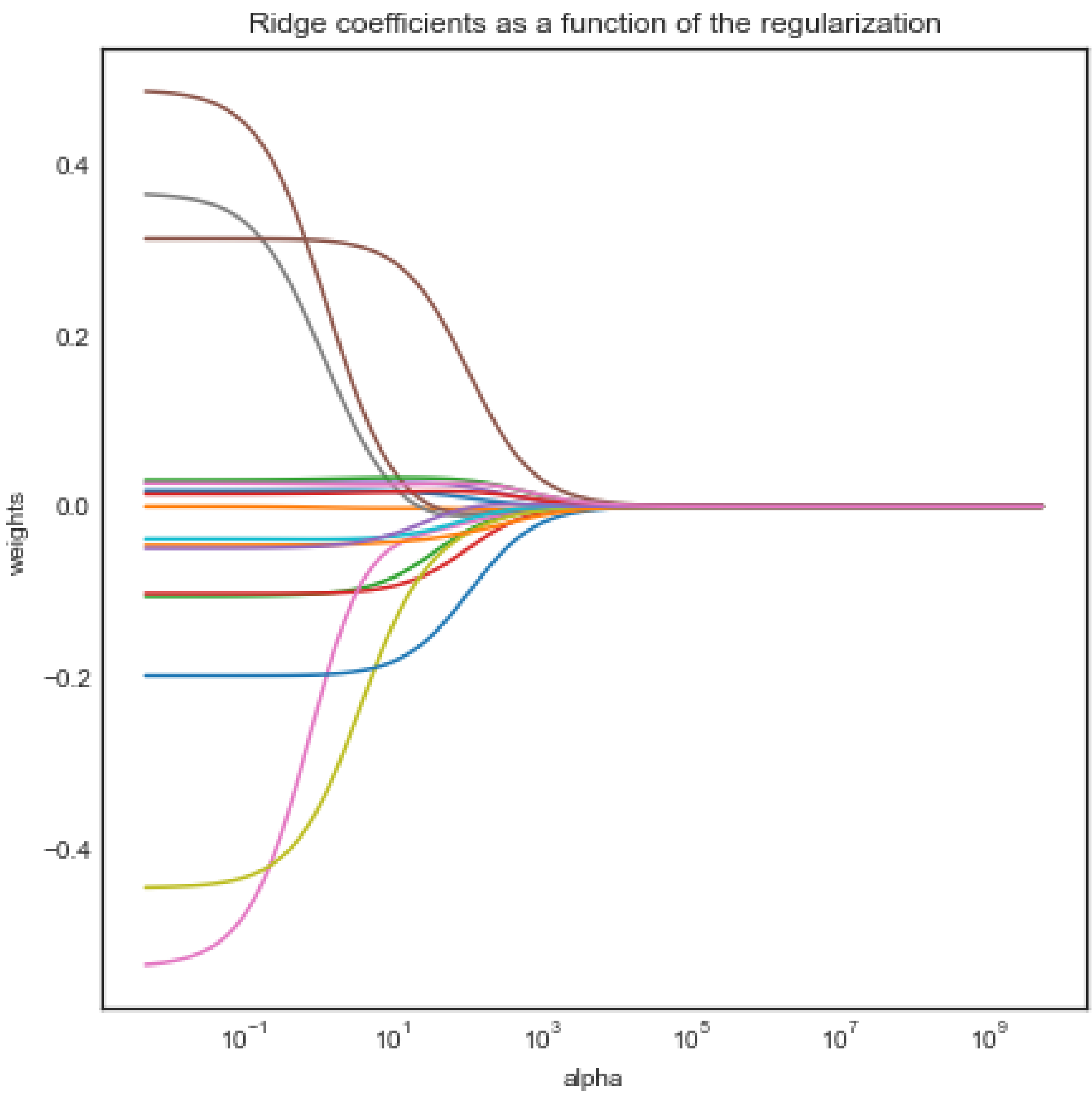


This correlation plot allows us to visually identify all relationships between variables, ranging all the way from -0.92 (close to perfectly inverse) to 1, indicating several perfect connections variables.

[43]:
```python
# data cleaning
```

```
data = data.dropna()                                                     #␣
 ↪506*19 - 494*19
dummies = pd.get_dummies(data[['3D_available']])                         #␣
 ↪494*2 - 1 or 0
data = data.drop(['3D_available'], axis=1)                               #␣
 ↪494*18 - no 3D YES/NO
data = pd.concat([data, dummies[['3D_available_YES']]], axis=1)          #␣
 ↪494*19 - 3D 1 or 0

# split X and Y
newX = data.drop(['Start_Tech_Oscar', 'Genre'], axis = 1)                #␣
 ↪494*17
newY = data['Start_Tech_Oscar']                                          #␣
 ↪494*1

# split train and test
X_train,X_test,y_train,y_test = sklearn.model_selection.train_test_split(
    newX, newY, test_size=0.3, random_state = 10)
```

[44]:
```
#Generate different values of alpha to fit different Ridge models
alphas = 10**np.linspace(10,-2,100)*0.5

ridge = Ridge()
coefs = []

for a in alphas:
    ridge.set_params(alpha=a)
    ridge.fit(scale(newX), newY)
    coefs.append(ridge.coef_)

plt.figure(figsize=(7, 7))
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
#ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization');
```



[45]:
```
scaler = StandardScaler().fit(X_train)
```

[46]:
```
# Perform CV and figure out the best alpha
ridgecv = RidgeCV(alphas=alphas, scoring='neg_mean_squared_error')
ridgecv.fit(scale(X_train), y_train)
print('Ridge alpha =', ridgecv.alpha_)
```

```
Ridge alpha = 0.4348745013088917
```

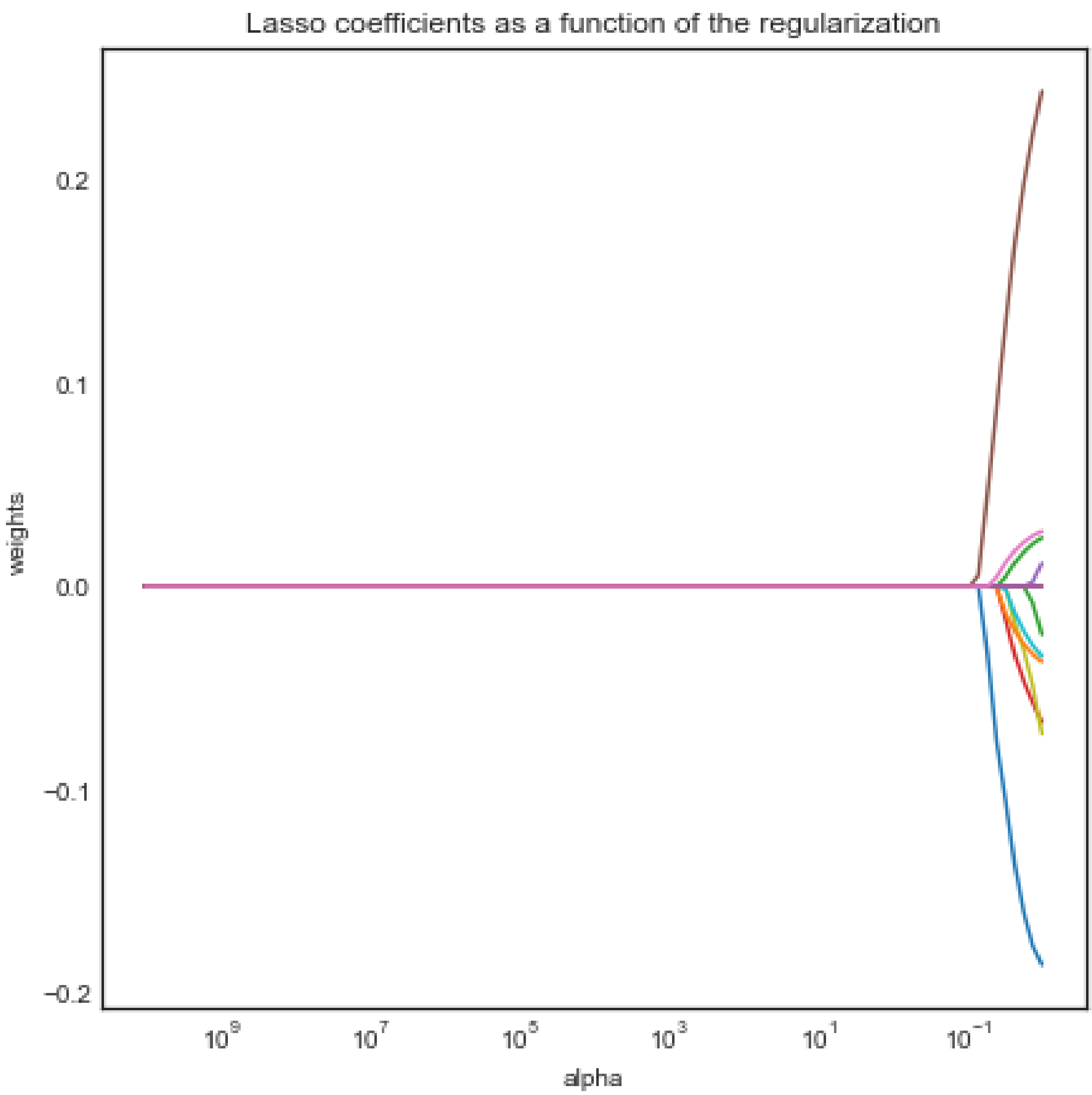
[47]:
```
# Estimate the Ridge Model using the best alpha
ridge2 = Ridge()
ridge2.set_params(alpha=0.4348745013088917)
ridge2.fit(scale(X_train), y_train)
```

```
mean_squared_error(y_test, ridge2.predict(scale(X_test)))
print('MSE = ', mean_squared_error(y_test, ridge2.predict(scale(X_test))))
pd.Series(ridge2.coef_.flatten(), index=newX.columns)
```

```
MSE =  0.22884003850357437
```

[47]:
```
Marketing expense      0.003978
Production expense    -0.001698
Multiplex coverage    -0.074519
Budget                -0.093257
Movie_length           0.036134
Lead_ Actor_Rating     0.682304
Lead_Actress_rating   -0.480387
Director_rating        0.251312
Producer_rating       -0.583666
Critic_rating         -0.045632
Trailer_views         -0.214620
Time_taken            -0.050685
Twitter_hastags        0.028511
Avg_age_actors         0.003928
Num_multiplex         -0.018708
Collection             0.300376
3D_available_YES       0.038242
dtype: float64
```

[48]:
```
#LASSO
# Generate different values of alpha to fit different Ridge models
lasso = Lasso(max_iter=10000)
coefs = []

for a in alphas*2:
    lasso.set_params(alpha=a)
    lasso.fit(scale(X_train), y_train)
    coefs.append(lasso.coef_)
plt.figure(figsize=(7, 7))
ax = plt.gca()
ax.plot(alphas*2, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Lasso coefficients as a function of the regularization');
```



[49]:
```
lassocv = LassoCV(alphas=None, cv=10, max_iter=10000)
lassocv.fit(scale(X_train), y_train.values.ravel())
```

[49]:
```
LassoCV(cv=10, max_iter=10000)
```

[50]:
```
lassocv.alpha_
```

[50]:
```
0.0017531554062430593
```

[51]:
```
pd.Series(lasso.coef_, index=newX.columns)
```

[51]:
```
Marketing expense     -0.000000
Production expense    -0.000000
```

```
Multiplex coverage    -0.023622
Budget                -0.066727
Movie_length           0.011344
Lead_ Actor_Rating    -0.000000
Lead_Actress_rating   -0.000000
Director_rating       -0.000000
Producer_rating       -0.072485
Critic_rating         -0.034419
Trailer_views         -0.186127
Time_taken            -0.037044
Twitter_hastags        0.023906
Avg_age_actors         0.000000
Num_multiplex          0.000000
Collection             0.243543
3D_available_YES       0.027096
dtype: float64
```

[52]:
```python
lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(scale(X_train), y_train)
print('MSE = ',mean_squared_error(y_test, lasso.predict(scale(X_test))))
```

```
MSE =  0.22532078753761933
```

[53]:
```python
# Step 1: Use CV to get the best alpha
enetcv = ElasticNetCV(cv=10, max_iter=10000)
enetcv.fit(scale(X_train), y_train.values.ravel())

# Step 2: Estimate the model w/ best alpha
enet_best = ElasticNet(alpha=enetcv.alpha_)
enet_best.fit(scale(X_train), y_train)

# Step 3: Print model estimates
print(list(zip(enet_best.coef_, newX)))

# Step 4: Print Error Metrics

print('MSE',mean_squared_error(y_test, enetcv.predict(scale(X_test))))
```

```
[(0.003682700664003383, 'Marketing expense'), (0.00014356385080630177,
'Production expense'), (-0.07060803189691985, 'Multiplex coverage'),
(-0.09147593761157329, 'Budget'), (0.036190123531168815, 'Movie_length'),
(0.8817014400727423, 'Lead_ Actor_Rating'), (-0.6586712782319618,
'Lead_Actress_rating'), (0.28172629499553775, 'Director_rating'),
(-0.6357448733243666, 'Producer_rating'), (-0.044269891266599734,
'Critic_rating'), (-0.2151420592866737, 'Trailer_views'),
(-0.050520714804323213, 'Time_taken'), (0.027043885410953518,
'Twitter_hastags'), (0.003844124090292974, 'Avg_age_actors'),
(-0.01566769360198046, 'Num_multiplex'), (0.29911219999780425, 'Collection'),
```

```
(0.038982620574758035, '3D_available_YES')]
MSE 0.23172439193909705

C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:530: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.8601542419681394, tolerance: 0.00857217391304348
  model = cd_fast.enet_coordinate_descent(
```

[54]:
```
#PCA

pca = PCA()
X_reduced = pca.fit_transform(scale(newX))

print(pca.components_.shape)  #   Loadings
pd.DataFrame(pca.components_.T).loc[:4,:5]
```

```
(17, 17)
```

[54]:

	0	1	2	3	4	5
0	-0.185507	-0.152768	-0.130542	0.006585	0.014600	0.090904
1	-0.305953	-0.060285	-0.020739	0.005839	-0.045047	0.000198
2	0.323970	-0.031849	-0.046057	0.016789	-0.009430	0.164824
3	0.154522	0.495704	-0.018100	0.053622	0.097453	0.016320
4	-0.301522	0.061712	0.042083	-0.040495	0.094484	0.007149

[55]:
```
print(X_reduced.shape) #   Principal Components
pd.DataFrame(X_reduced).loc[:4,:5]
```

```
(494, 17)
```

[55]:

	0	1	2	3	4	5
0	1.329772	0.758568	2.091228	0.017536	0.202127	0.458976
1	1.419002	-0.595590	-0.339652	0.319339	-0.155415	-0.096741
2	2.221437	1.216727	-3.464258	13.386820	8.751526	0.683249
3	3.332915	0.917429	-0.576504	-0.912837	0.175936	-0.065317
4	3.517164	0.778602	-1.502983	-0.131155	-0.392351	-1.131371

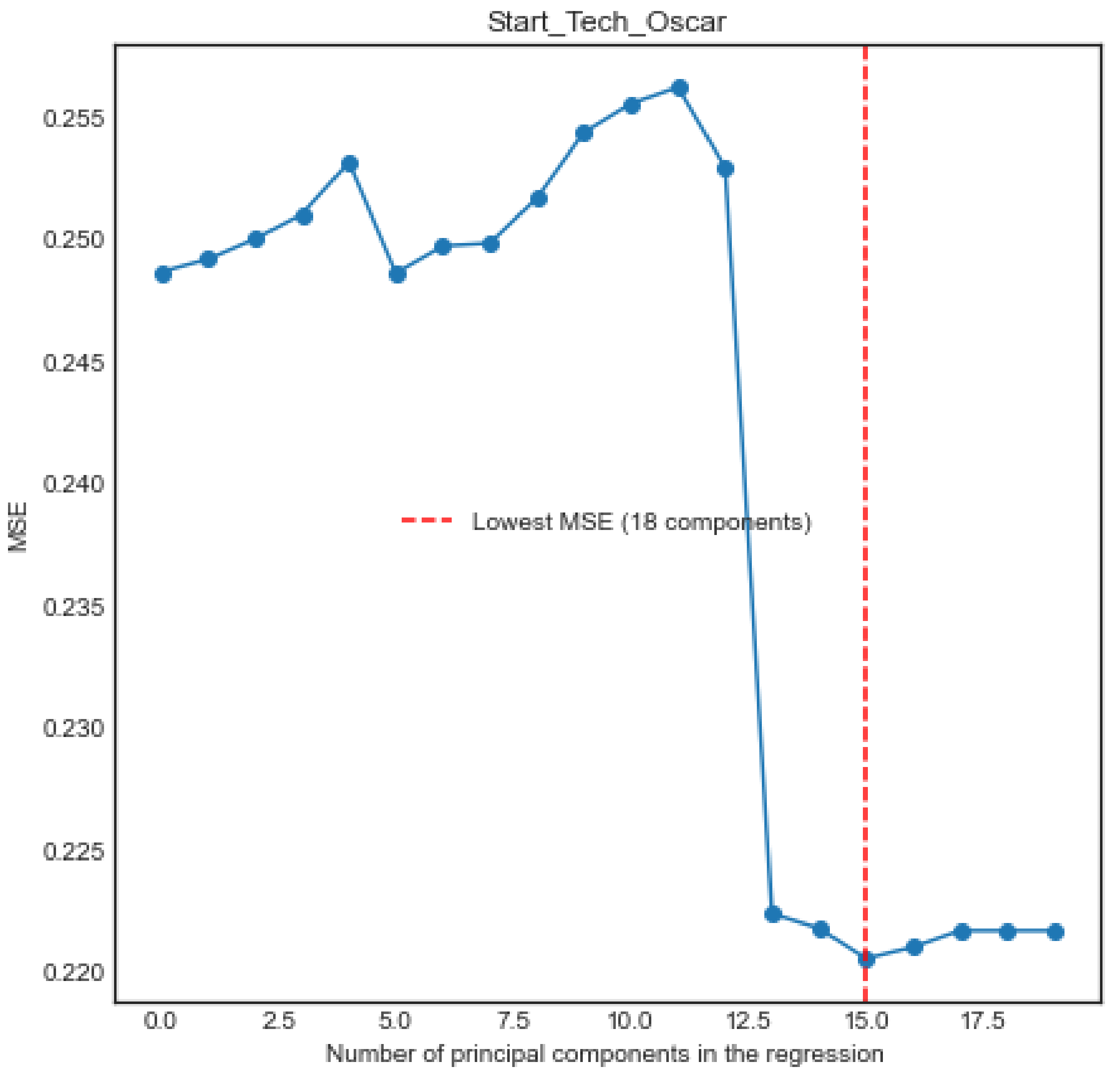
[56]:
```
np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

[56]:
```
array([ 44.67,  56.5 ,  62.99,  69.17,  74.78,  80.34,  85.14,  89.56,
        92.84,  95.16,  96.83,  98.23,  99.49,  99.94,  99.99, 100.  ,
       100.01])
```

[57]:
```
# 10-fold CV, with shuffle
n = len(X_reduced)
kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)

regr = LinearRegression()
```

```
mse = []

# Calculate MSE with only the intercept (no principal components in regression)
score = -1*cross_val_score(regr, np.ones((n,1)), newY.ravel(), cv=kf_10,␣
↪scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using CV for the 19 principle components, adding one component␣
↪at the time.
for i in np.arange(1, 20):
    score = -1*cross_val_score(regr, X_reduced[:,:i], newY.ravel(), cv=kf_10,␣
↪scoring='neg_mean_squared_error').mean()
    mse.append(score)
plt.figure(figsize=(7, 7))
plt.plot(mse, '-o')
plt.xlabel('Number of principal components in the regression')
plt.ylabel('MSE')
plt.title('Start_Tech_Oscar')
plt.xlim(xmin=-1);
plt.axvline(15, linestyle="--", color="r", label="Lowest MSE (18 components)")
plt.legend(loc='center')
```

[57]: <matplotlib.legend.Legend at 0x21bb4d60b50>

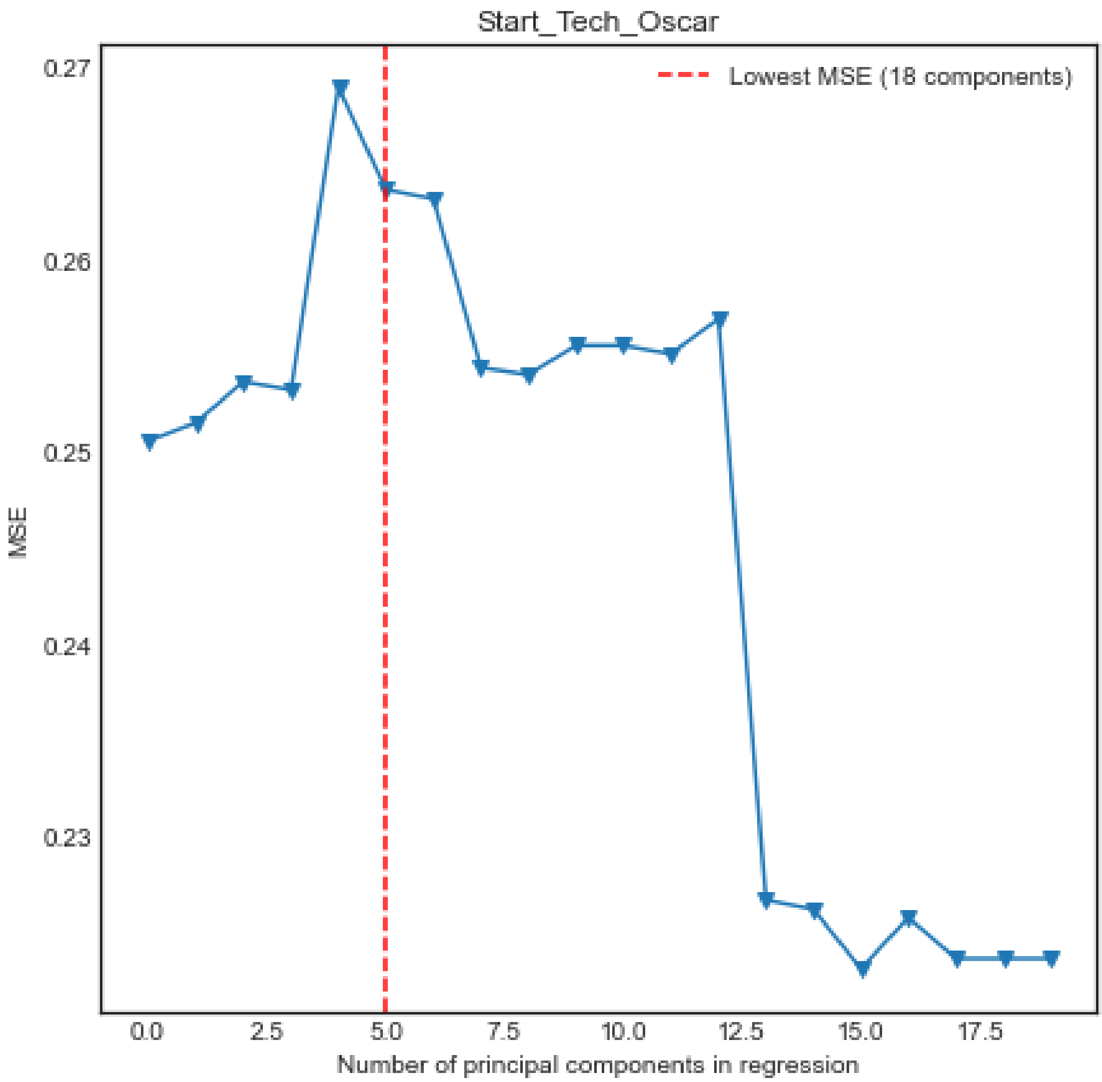


[58]:
```
regr_test = LinearRegression()
regr_test.fit(X_reduced, newY)
regr_test.coef_
```

[58]:
```
array([ 1.87228106e-03,  8.11239933e-03,  1.78914460e-02,  2.13907617e-03,
        6.81624346e-02, -1.81636205e-02,  3.02496952e-02, -4.01603354e-04,
       -3.69052854e-03, -3.09438240e-02, -2.08973018e-02,  1.22460914e-01,
        3.62964344e-01,  1.42292123e-01, -4.60900244e-01,  3.61470596e-01,
       -7.17305902e-01])
```

After running CV shuffle for PCA, we found out that 15 components results in the lowest MSE; however, 15 components is high considering we have 18 variables. We will run different types of PCA to determine the optimal number of components.

[59]:
```python
pca2 = PCA()
X_reduced_train = pca2.fit_transform(scale(X_train))
n = len(X_reduced_train)

# 10-fold CV, with shuffle
kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)

mse = []

# Calculate MSE with only the intercept (no principal components in regression)
score = -1*cross_val_score(regr, np.ones((n,1)), y_train, cv=kf_10,␣
↪scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using CV for the 19 PCs, adding one component at the time.
for i in np.arange(1, 20):
    score = -1*cross_val_score(regr, X_reduced_train[:,:i], y_train, cv=kf_10,␣
↪scoring='neg_mean_squared_error').mean()
    mse.append(score)
plt.figure(figsize=(7, 7))
plt.plot(np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Start_Tech_Oscar')
plt.axvline(5, linestyle="--", color="r", label="Lowest MSE (18 components)")
plt.legend(loc='best')
plt.xlim(xmin=-1);
```



[60]:
```
X_reduced_test = pca2.transform(scale(X_test))[:,:7]

# Train regression model on training data
regr = LinearRegression()
regr.fit(X_reduced_train[:,:7], y_train)

# Prediction with test data
pred = regr.predict(X_reduced_test)
mean_squared_error(y_test, pred)
```

[60]: 0.24363911782518424

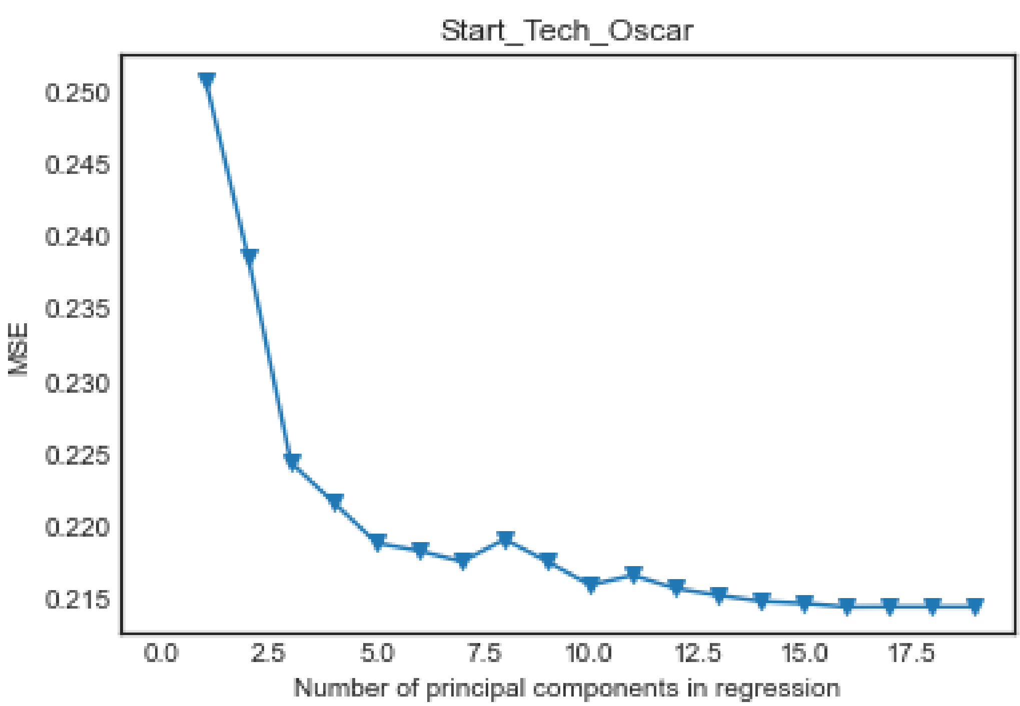
After running a PCA using training data, our model still indicates that 15 components will result in the minimal MSE despite being a large number of components.

[61]:
```
n = len(X_train)

# 10-fold CV, with shuffle
kf_10 = KFold(n_splits=10, shuffle=True, random_state=0)

mse = []

for i in np.arange(1, 20):
    pls = PLSRegression(n_components=i)
    score = cross_val_score(pls, scale(X_train), y_train, cv=kf_10,␣
 ↪scoring='neg_mean_squared_error').mean()
    mse.append(-score)

plt.plot(np.arange(1, 20), np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Start_Tech_Oscar')
plt.xlim(xmin=-1);
```

```
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
```

```
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(18) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
```

```
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
C:\Users\Zachary DeBar\anaconda3\lib\site-
packages\sklearn\cross_decomposition\_pls.py:206: FutureWarning: As of version
0.24, n_components(19) should be in [1, n_features].n_components=17 will be used
instead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
```



[62]:
```
pls = PLSRegression(n_components=2)
pls.fit(scale(X_train), y_train)

mean_squared_error(y_test, pls.predict(scale(X_test)))
```

[62]: 0.2340087208422181

After running a PLS, MSE is still lower as the number of components increases; however, unlike the previous PCA results which required 13+ components to significantly lower MSE, 5-7 components yielded an MSE fairly close to the minimum possible MSE. This implies that we can use fewer components without greatly sacrificing our model's performance.

0.2.1 Regularization Conclusion

The Lasso regression yielded the lowest MSE of 0.225 from the models we've created, and it was able to successfully shrink 7 coefficients to exactly 0.