Task1

February 20, 2019

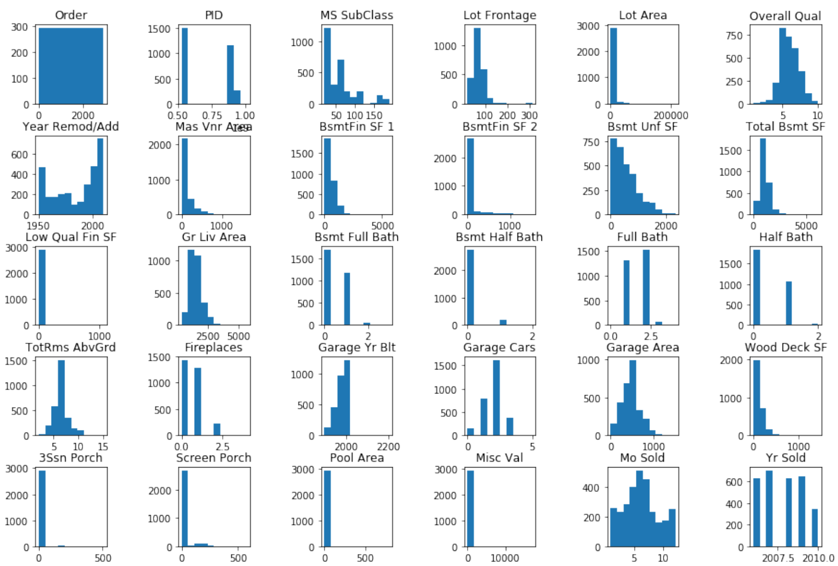
```
In [1]: import warnings
        warnings.filterwarnings('ignore')
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn import linear_model,  pipeline, preprocessing
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import OneHotEncoder
        from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import cross_validate
        from sklearn.preprocessing import StandardScaler
        from sklearn.compose import ColumnTransformer
        from sklearn.impute import SimpleImputer
        from sklearn.linear_model import Ridge
        from sklearn.pipeline import Pipeline
        from heapq import nlargest
        from sklearn.datasets import make_regression
        from sklearn.linear_model import ElasticNet
        from sklearn.pipeline import make_pipeline
        from sklearn.model_selection import GridSearchCV
        from sklearn.preprocessing import PolynomialFeatures, scale
        from sklearn.exceptions import DataConversionWarning
        warnings.filterwarnings(action='ignore', category=DataConversionWarning)
```

1 1.0: Load Data

```
In [2]: data = pd.read_excel("AmesHousing.xls")
        target = data['SalePrice']
        data= data.drop(['SalePrice'], axis=1)
        df = data.select_dtypes(exclude=['object'])
        categorical = data.select_dtypes(exclude=['int64','float64'])
        categorical_name = list(categorical)
        numeric_features = list(df)
```

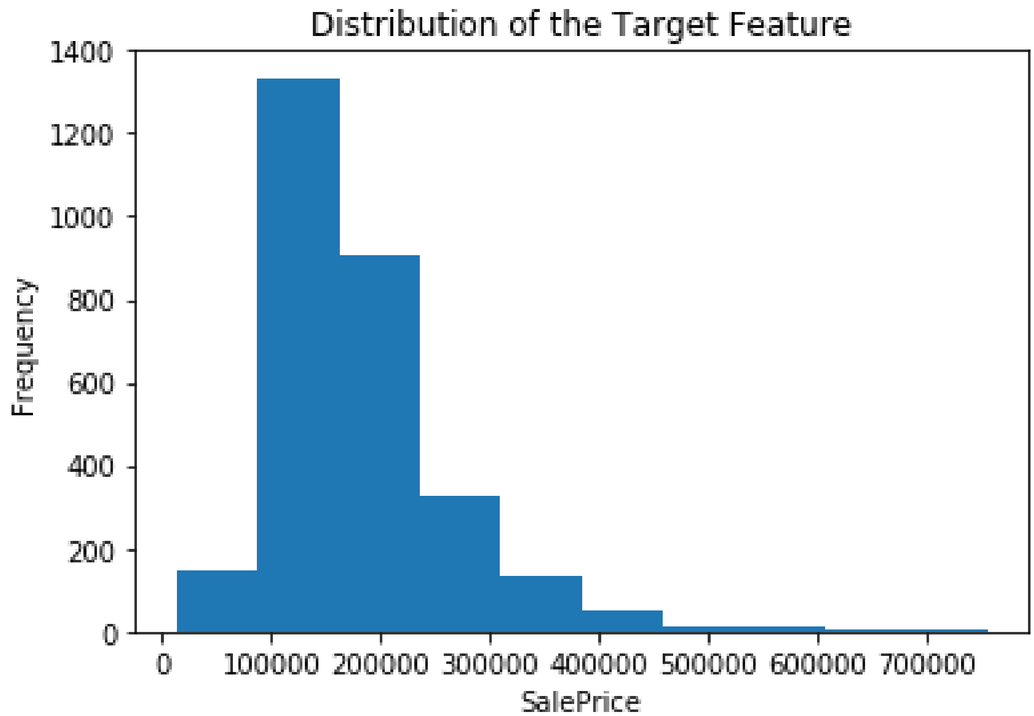
2 1.1: Visualize the Univariate Distribution of Each Continuous, and the Distribution of the Target

In [3]:
```
# Create histogram for each univariate distribution
fig = plt.figure(figsize=(20,10))
fig.subplots_adjust(hspace=0.4, wspace=1)
for i in range(len(df.columns)):
    ax = fig.add_subplot(5, 8, i+1)
    ax.hist(df.iloc[:,i])
    ax.set_title(list(df)[i])
```



In [4]:
```
#Visualize the target feature
plt.hist(target)
plt.title("Distribution of the Target Feature")
plt.ylabel("Frequency")
plt.xlabel("SalePrice")
```

Out[4]:
```
Text(0.5, 0, 'SalePrice')
```

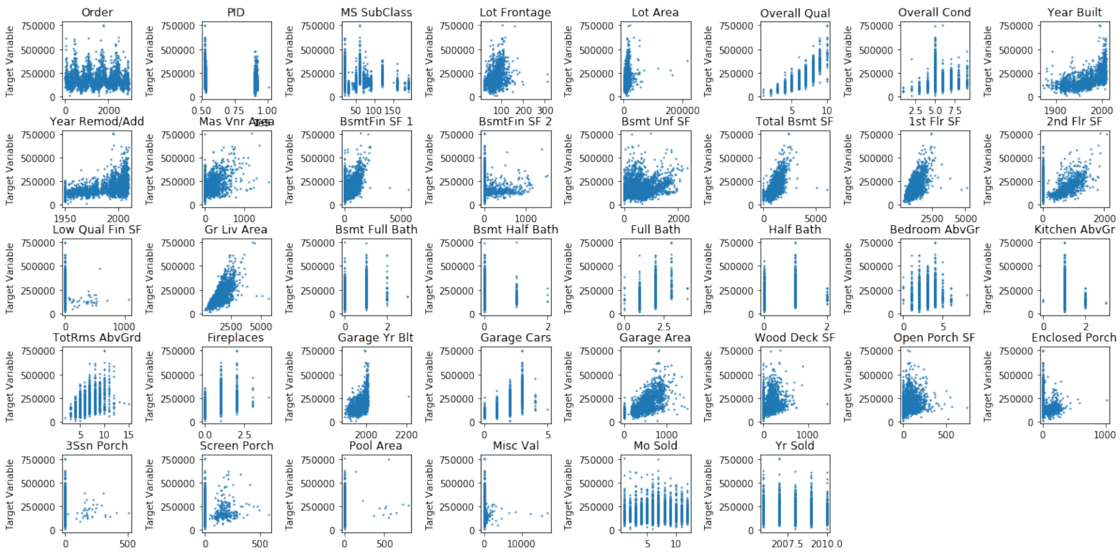


3 Notice & Treatment for 1.1

There are some continuous features following the Gaussian distribution. However, most variables are heavily skewed. Also, "Other" seems to follow the uniform distribution. For those skewed distributions, it would be better to take a log transformation to make them more normal distribution shape.

4 1.2: Visualize the Dependency between the Target Feature and Each Continuous Variables

```
In [5]: # Create subplots and plot scatter plots on each continuous features with respect to th
        fig = plt.figure(figsize=(20,10))
        fig.subplots_adjust(hspace=0.4, wspace=1)
        for i in range(len(df.columns)):
            ax = fig.add_subplot(5, 8, i+1)
            ax.scatter(df.iloc[:,i],target,s=1)
            ax.set_title(list(df)[i])
            ax.set_ylabel("Target Variable")
```



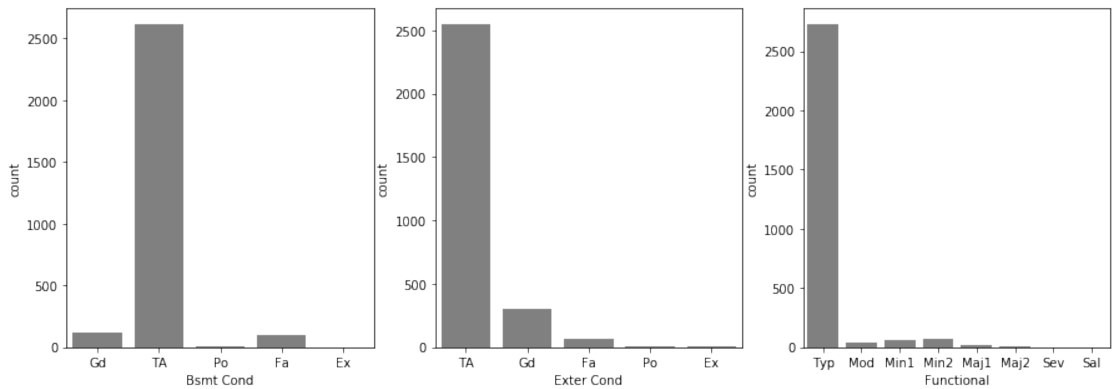
5 1.3: Split Data & Cross Validate the models & Visualize the Relationships among the Top Three R Squared Features

In [6]:
```
# Look for the top three R^2 features
R_2_dic = {}
for i in range(len(categorical_name)):
    dummy_categorical = pd.get_dummies(categorical.iloc[:,i-1])

    X_train, X_test, y_train, y_test = train_test_split(dummy_categorical, target, tes

    scaler = StandardScaler()
    md = scaler.fit(X_train)
    X_train_scaled = md.transform(X_train)


    lm = linear_model.LinearRegression()
    model = lm.fit(X_train_scaled,y_train)
    scores = cross_val_score(model,dummy_categorical,target, cv=3, scoring='r2')
    avg_scores = sum(scores) / float(len(scores))
    R_2_dic.update({categorical_name[i]:avg_scores})




sorted_dic = sorted(R_2_dic.items(), key=lambda v: v[1], reverse=True)
print(sorted_dic[0:3])
```

```
[('Bsmt Cond', 0.5088519401667361), ('Exter Cond', 0.4977580547800917), ('Functional', 0.47578
```

In [7]:
```
# Store the top three R^2 features into another dataframe
cat = categorical[['Bsmt Cond','Exter Cond','Functional']]

# Plot histgram correspoinding to those features
f, axes = plt.subplots(1, 3,figsize=(15,5))
sns.countplot(cat['Bsmt Cond'], color='grey', ax=axes[0])
sns.countplot(cat['Exter Cond'], color='grey', ax=axes[1])
sns.countplot(cat['Functional'], color='grey', ax=axes[2])
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1a25813b00>



6 1.4: Evaluation on the Models

In [8]:
```
# Create a categorical transformer
categorical_features = categorical_name
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Take a column transfomer
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', categorical_transformer, categorical_features)])

# Create pipelines for each model
reg = Pipeline(steps=[('preprocessor', preprocessor),
                      ('ols', linear_model.LinearRegression())])

ridge = Pipeline(steps=[('preprocessor', preprocessor),
                        ('ridge', Ridge())])
```

```
lasso = Pipeline(steps=[('preprocessor', preprocessor),
                      ('lasso', linear_model.Lasso(tol=1))])

elastic = Pipeline(steps=[('preprocessor', preprocessor),
                      ('elastic', ElasticNet())])
```

In [9]:
```
#Split data into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(categorical, target, test_size=0.3

#Fit each model
reg.fit(X_train,y_train)
scores = cross_val_score(reg,X_train, y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("Regression:")
print(scores)
print("")

ridge.fit(X_train,y_train)
scores = cross_val_score(ridge,X_train, y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("Ridge:")
print(scores)
print("")

lasso.fit(X_train,y_train)
scores = cross_val_score(lasso,X_train, y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("Lasso:")
print(scores)
print("")

elastic.fit(X_train,y_train)
scores = cross_val_score(elastic,X_train, y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("ElasticNet:")
print(scores)
```

```
Regression:
0.788688062216835

Ridge:
0.8075366230619507

Lasso:
0.7037713902695856

ElasticNet:
0.65004956062464
```

In [10]:
```
# Create pipelines to transform categorical and continuous features
numeric_transformer = Pipeline(steps=[
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant',fill_value = 'missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num',numeric_transformer, numeric_features),
        ('cat', categorical_transformer,categorical_features)])


# Instantiate classifiers for each model
clf1 = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('ridge', Ridge())])

clf2 = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('lasso', linear_model.Lasso(tol=1))])

clf3 = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('elastic', ElasticNet())])
```

In [11]:
```
#Split data into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.3)

#Fit each model and evaluate cross validation scores
clf1.fit(X_train,y_train)
scores = cross_val_score(clf1,X_train,y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("Ridge After Tuning the Parameter:")
print(scores)
print("")

clf2.fit(X_train,y_train)
scores = cross_val_score(clf2,X_train,y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("Lasso After Tuning the Parameter:")
print(scores)
print("")
```

```
clf3.fit(X_train,y_train)
scores = cross_val_score(clf3,X_train,y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("Elastic After Tuning the Parameter:")
print(scores)
```

```
Ridge After Tuning the Parameter:
0.8784526910292412

Lasso After Tuning the Parameter:
0.7684014950067057

Elastic After Tuning the Parameter:
0.8459978082350457
```

7 Conclusion for 1.4:

Yes, scaling helps the model significantly improve the result.

8 1.5: Tune the Parameter by the GridSearchCV

In [12]:
```
# Split dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.3)

# Seach for the best parameter by GridSearchCV for Ridge Regression
param_grid = {'ridge__alpha': [0.01,0.1, 1, 10, 100]}

ridge_grid = GridSearchCV(clf1, param_grid=param_grid, cv=5)
ridge_grid.fit(X_train, y_train)


print("best mean cross-validation score of Ridge: {:.3f}".format(ridge_grid.best_score
print("best parameters of Ridge: {}".format(ridge_grid.best_params_))
print("test-set score of Ridge: {:.3f}".format(ridge_grid.score(X_test, y_test)))
print("")


# Seach for the best parameter by GridSearchCV for Lasso Regression
param_grid = {'lasso__alpha': np.logspace(-3, 0, 13)}

lasso_grid = GridSearchCV(clf2, param_grid=param_grid, cv=5)
lasso_grid.fit(X_train, y_train)

print("best mean cross-validation score of Lasso: {:.3f}".format(lasso_grid.best_score
print("best parameters of Lasso: {} ".format(lasso_grid.best_params_))
print("test-set score of Lasso: {:.3f}".format(lasso_grid.score(X_test, y_test)))
```

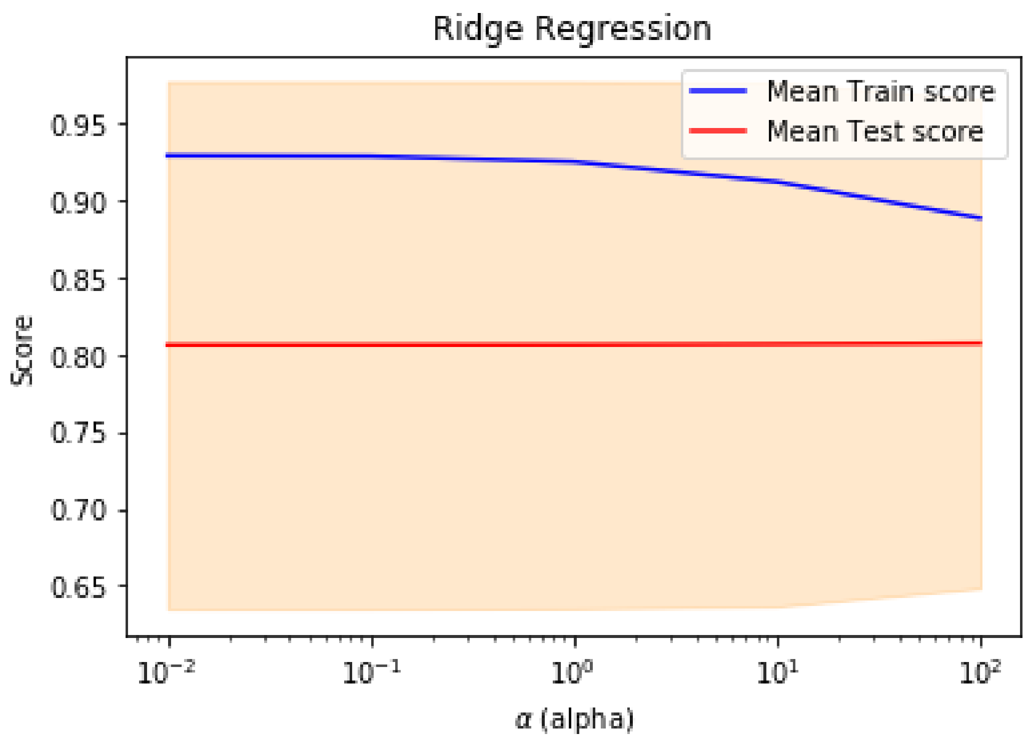
```
    print("")


    # Seach for the best parameter by GridSearchCV for ElastcNet
    param_grid = {'elastic__alpha': [0.01,0.1, 1, 10, 100]}

    elastic_grid = GridSearchCV(clf3, param_grid=param_grid, cv=5)
    elastic_grid.fit(X_train, y_train)

    print("best mean cross-validation score of Elastic: {:.3f}".format(elastic_grid.best_s
    print("best parameters of Elastic: {}".format(elastic_grid.best_params_))
    print("test-set score of Elastic: {:.3f}".format(elastic_grid.score(X_test, y_test)))
```

```
best mean cross-validation score of Ridge: 0.807
best parameters of Ridge: {'ridge__alpha': 100}
test-set score of Ridge: 0.912

best mean cross-validation score of Lasso: 0.708
best parameters of Lasso: {'lasso__alpha': 1.0}
test-set score of Lasso: 0.813



/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/coordinate_descent.py:492: Converge
  ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/coordinate_descent.py:492: Converge
  ConvergenceWarning)


best mean cross-validation score of Elastic: 0.807
best parameters of Elastic: {'elastic__alpha': 0.1}
test-set score of Elastic: 0.912
```

```
In [13]: # Visualize the dependence of the validation score for Ridge Regression
         ridge_alphas = [0.01,0.1, 1, 10, 100]

         train_scores_mean = ridge_grid.cv_results_["mean_train_score"]
         train_scores_std = ridge_grid.cv_results_["std_train_score"]
         test_scores_mean = ridge_grid.cv_results_["mean_test_score"]
         test_scores_std = ridge_grid.cv_results_["std_test_score"]

         plt.figure()
         plt.title('Ridge Regression')
         plt.xlabel('$\\alpha$ (alpha)')
         plt.ylabel('Score')

         plt.semilogx(ridge_alphas, train_scores_mean, label='Mean Train score',
                      color='blue')
```

```
plt.semilogx(ridge_alphas, test_scores_mean,
             label='Mean Test score', color='red')


plt.gca().fill_between(ridge_alphas,
                       test_scores_mean - test_scores_std,
                       test_scores_mean + test_scores_std,
                       alpha=0.2,
                       color='darkorange')

plt.legend(loc='best')
```

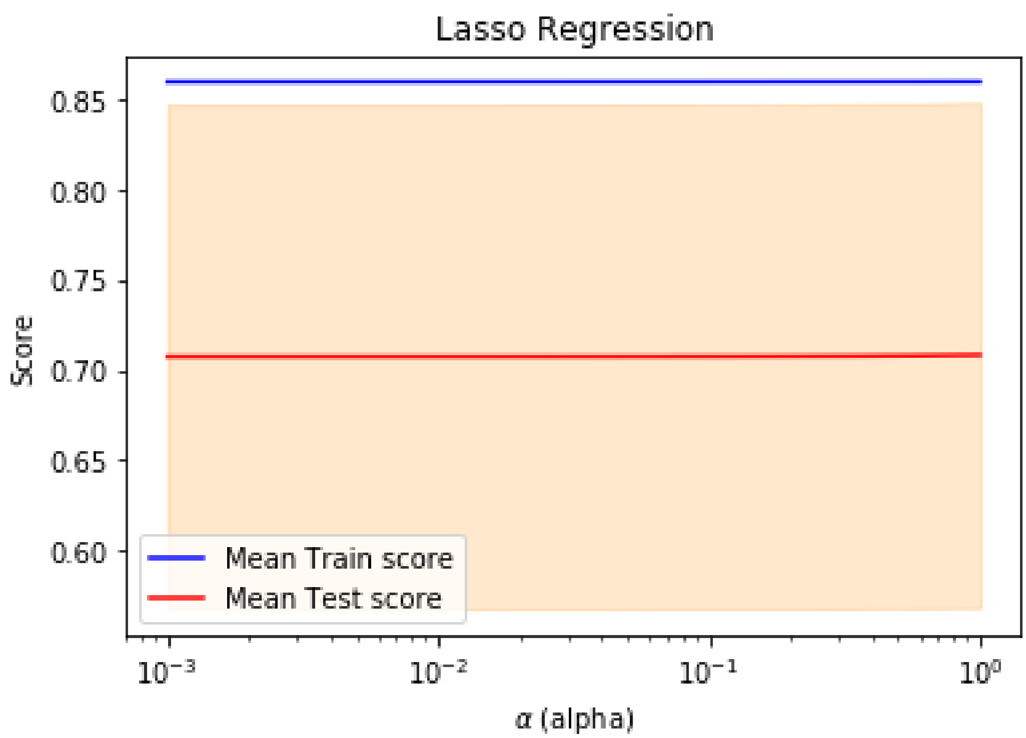
Out[13]: <matplotlib.legend.Legend at 0x1a25d794a8>



In [15]:
```
# Visualize the dependence of the validation score for Lasso Regression
lasso_alphas = np.logspace(-3, 0, 13)

train_scores_mean = lasso_grid.cv_results_["mean_train_score"]
train_scores_std = lasso_grid.cv_results_["std_train_score"]
test_scores_mean = lasso_grid.cv_results_["mean_test_score"]
test_scores_std = lasso_grid.cv_results_["std_test_score"]
```

```
plt.figure()
plt.title('Lasso Regression')
plt.xlabel('$\\alpha$ (alpha)')
plt.ylabel('Score')

plt.semilogx(lasso_alphas, train_scores_mean, label='Mean Train score',
             color='blue')
plt.semilogx(lasso_alphas, test_scores_mean,
             label='Mean Test score', color='red')


plt.gca().fill_between(lasso_alphas,
                       test_scores_mean - test_scores_std,
                       test_scores_mean + test_scores_std,
                       alpha=0.2,
                       color='darkorange')

plt.legend(loc='best')
```

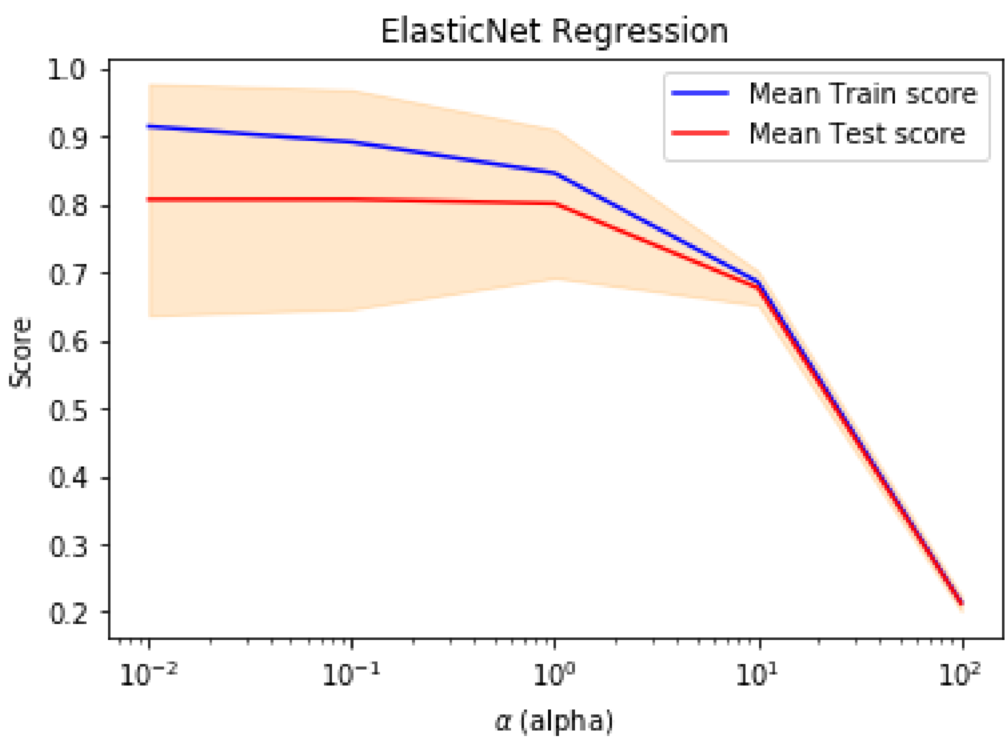
Out[15]: <matplotlib.legend.Legend at 0x1a25e64cc0>




```
In [16]: # Visualize the dependence of the validation score for ElasticNet
         elastic_alphas = [0.01,0.1, 1, 10, 100]
```

```
train_scores_mean = elastic_grid.cv_results_["mean_train_score"]
train_scores_std = elastic_grid.cv_results_["std_train_score"]
test_scores_mean = elastic_grid.cv_results_["mean_test_score"]
test_scores_std = elastic_grid.cv_results_["std_test_score"]

plt.figure()
plt.title('ElasticNet Regression')
plt.xlabel('$\\alpha$ (alpha)')
plt.ylabel('Score')
# plot train scores
plt.semilogx(elastic_alphas, train_scores_mean, label='Mean Train score',
             color='blue')
plt.semilogx(elastic_alphas, test_scores_mean,
             label='Mean Test score', color='red')

# create a shaded area between [mean - std, mean + std]
plt.gca().fill_between(elastic_alphas,
                       test_scores_mean - test_scores_std,
                       test_scores_mean + test_scores_std,
                       alpha=0.2,
                       color='darkorange')

plt.legend(loc='best')
```

Out[16]: <matplotlib.legend.Legend at 0x10ff72908>



9 Conclusion for 1.5:

Yes, tuning the parameter increases the results for all models.

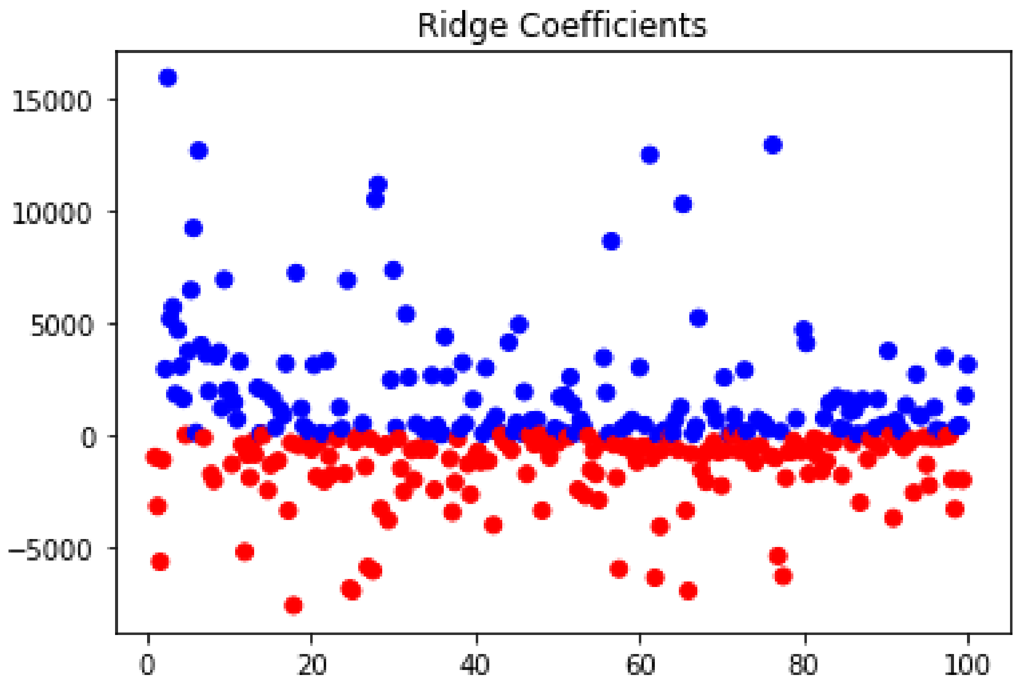
10 1.6: Visualize the Coefficients

In [17]:
```
# Length of Ridge coefficients
len(ridge_grid.best_estimator_.steps[1][1].coef_)
```

Out[17]:
```
318
```

In [18]:
```
# Visualize the coefficients of Ridge Regression
ridge_clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('ridge', Ridge(alpha=0.01))])

ridge = ridge_clf.fit(X_train,y_train)

plt.scatter(np.linspace(1,100,318),ridge_grid.best_estimator_.steps[1][1].coef_,c=np.s
plt.title('Ridge Coefficients')
#
```

Out[18]:
```
Text(0.5, 1.0, 'Ridge Coefficients')
```



In [19]:
```
# Take argsort on the list containing all coefficients
np.argsort(np.absolute(ridge_grid.best_estimator_.steps[1][1].coef_))
```

Out[19]:
```
array([112, 277, 196, 274, 111, 150, 287, 177,  70, 210, 224, 128,  12,
        80, 135,  41, 225,  42,  65, 202, 146, 310, 143,  16, 164, 200,
       309, 169, 201, 134, 159, 229, 106, 244, 140,  60, 300, 231, 307,
       161,  79,  19, 291, 303, 306, 257, 199, 269, 279, 118,  95, 305,
       239,  84,  71, 182, 105,  73, 295, 227, 119, 156, 267, 259,  94,
       281,  47, 249, 155,  59,  78, 280, 136, 218, 265, 211,  38, 264,
       313,  58,  64, 174, 137, 290, 167, 110,  53, 157, 294, 314, 184,
       285, 191, 131, 272, 284,  90, 238, 102, 148, 121, 152, 179,  81,
        34, 237, 141, 236, 185,  56, 192, 190, 251,  89, 183, 153, 103,
       289, 233, 147, 292, 283, 219, 186, 166, 149, 203,  32, 104,  61,
       107, 235, 250, 256, 223, 198, 216, 204, 100, 139, 261, 171, 252,
       126, 228, 299, 133, 248, 247, 226, 220, 298,  50, 209, 187, 232,
        39,  36, 214, 258, 240,  68, 271, 125,   0, 154, 194, 242,  49,
       273, 115, 278,  57,   3,  26,  48, 304,  72, 217, 130, 205, 262,
       127, 188, 234, 293, 163, 122,  30, 301,  45, 263,  31,  82,  96,
       124, 270, 276, 282,  46,  11, 170, 213, 266, 260, 158, 160, 316,
        74, 255, 172, 145,   8,  22,  69, 268, 176, 144,  43,  63,  21,
        28,  37, 180, 246,  29, 311, 101, 315,  40,  23,  66, 215, 117,
       302, 221,  92, 165, 109,  44, 222,  99, 162,  97, 114, 296, 108,
       297, 123, 168, 230,   4, 173, 129, 189,  10, 275,  62, 317,  51,
       120,   1,  33,  67,  88, 312, 175, 207, 151,  52, 308,  24, 116,
        20,  25,  13, 286, 288,  91,  18, 132, 254, 138, 197, 113,   9,
       253, 142,   6, 212,  35,  98, 243,   7,   2,  83, 181,  85, 245,
       195,  14,  75,  76,  27, 208,  77,  55,  93,  54, 178,  15, 206,
        86,  87, 193,  17, 241,   5])
```

In [96]:
```
# Top three important categorical features
print("The first important featrue for Ridge Regression is: " + preprocessor.transform
print("The second important featrue for Ridge Regression is: " + preprocessor.transfo
print("The third important featrue for Ridge Regression is: " + preprocessor.transform
```

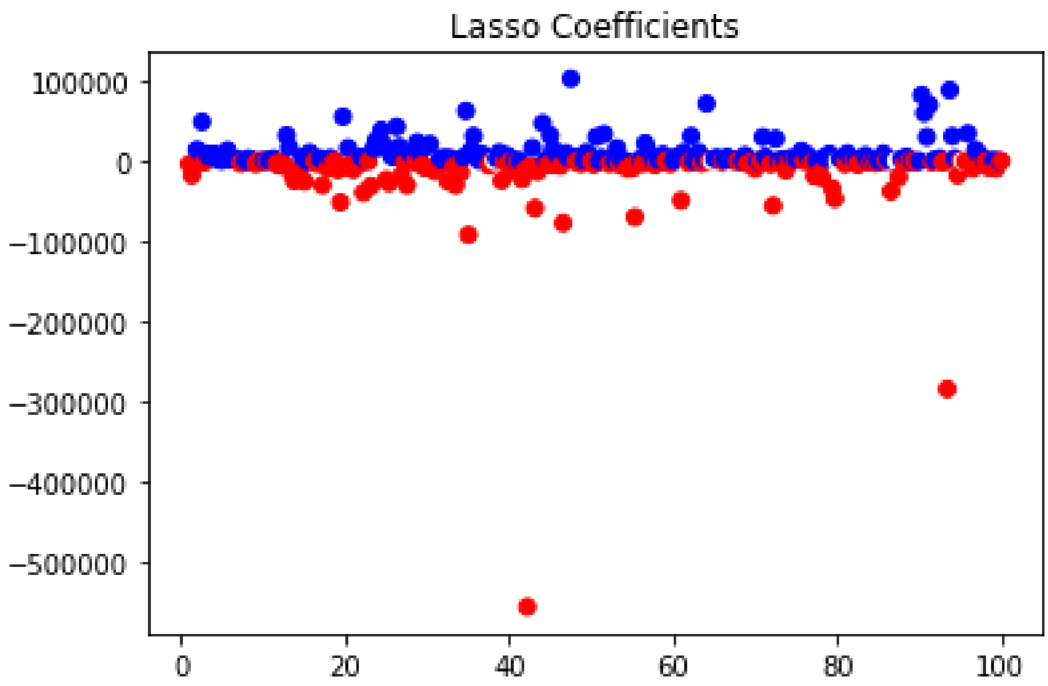
```
The first important featrue for Ridge Regression is: Bsmt Cond_Po
The second important featrue for Ridge Regression is: Land Contour_Lvl
The third important featrue for Ridge Regression is: Garage Type_Basment
The forth important featrue for Ridge Regression is: Condition 2_PosA
The fifth important featrue for Ridge Regression is: Bsmt Qual_missing
```

In [20]:
```
# Length of Ridge coefficients
len(lasso_grid.best_estimator_.steps[1][1].coef_)
```

Out[20]: 318

In [28]:
```
# Visualize the coefficients of Lasso Regression
lasso_clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('lasso', linear_model.Lasso(alpha=1.0,tol=1))])

lasso = lasso_clf.fit(X_train,y_train)

plt.scatter(np.linspace(1,100,318),lasso_grid.best_estimator_.steps[1][1].coef_,c=np.s
plt.title('Lasso Coefficients')
```

Out[28]: Text(0.5, 1.0, 'Lasso Coefficients')



In [29]:
```
# Take argsort on the list containing all coefficients
np.argsort(np.absolute(lasso_grid.best_estimator_.steps[1][1].coef_))
```

Out[29]:
```
array([253, 204, 276, 119, 302, 236,  46, 301,  27,  57, 282,  30, 275,
        49, 285, 186, 216, 281, 317, 272, 270, 116, 181, 157, 240, 203,
        53,  70, 267, 295, 316, 197,  24, 223, 245, 311, 152, 166, 303,
       160,  22, 283, 175, 222,  25, 226, 214,  48, 170, 305, 189,  65,
       215, 313, 129, 234, 269, 230,  29,  28, 198,  31, 265, 169, 208,
       118,  13, 113, 212, 125, 284, 151, 314, 128,  69, 259, 139,  32,
       127, 262,  12, 123, 154, 184,  19, 220, 263, 290,  36,   6, 291,
        88,  16, 209, 266,  99, 294, 163, 273, 292, 187, 213, 183, 260,
       176,  21,  84, 165, 243,  56, 195, 255, 180, 268, 232,  18,  11,
       238, 201,  17, 207, 278, 219, 237, 309, 200,  14, 279, 242, 205,
```

```
                    20, 254, 211,  23,  33,  34,  50, 231, 249, 293, 177,  98, 164,
                   218, 299, 120,  51, 188,  26, 153, 310, 102,   0, 256,  55, 227,
                   235, 258, 105, 161,  37, 148, 206, 158, 100,  66, 168, 261, 182,
                   107, 143, 140, 248, 191, 210, 190,  91, 225,  35,  97,  44, 117,
                   280,  67,   8,  80, 150, 145, 144,  79,  87,  63,   9, 137,  61,
                   194, 133,   2, 264, 124, 217, 115, 156, 312, 172, 114, 103,  92,
                   185, 112, 241, 173,  54,  10,   7, 250,  95, 171, 306, 315, 221,
                   271,  90, 257, 308, 147,  86,  47,  64, 121, 155, 199, 193,  58,
                    42, 233, 131,   4, 239,  96, 179,  43, 110, 136, 126,  15, 307,
                     3, 106, 246, 142,  40, 167,  72,  62, 134,  74,  39, 300,   1,
                    83,  82, 244,  93, 277,  76, 247,  94, 178, 130,  77,  89, 122,
                    41, 101,  45,  73,  78, 229, 224,  52, 104,  85, 288, 159,  71,
                   298, 196, 111, 141,  38, 162, 251, 304, 274,  75,  68,  81, 252,
                   138,   5, 192,  59,  60, 228, 135, 287, 108, 174, 289, 202, 146,
                   286, 297, 109, 149, 296, 132])
```

In [32]:
```python
# Top three important categorical features
print("The first important featrue for Lasso Regression is: " + preprocessor.transform
print("The first important featrue for Lasso Regression is: " + preprocessor.transform
print("The first important featrue for Lasso Regression is: " + preprocessor.transform
```

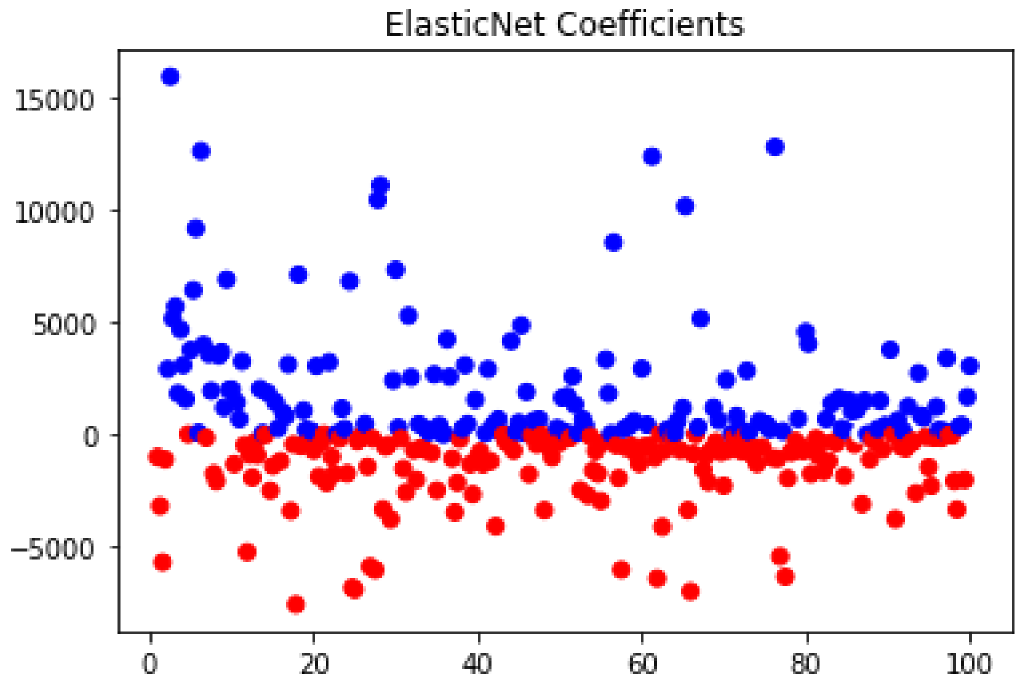
```
The first important featrue for Lasso Regression is: Fence_GdPrv
The first important featrue for Lasso Regression is: Kitchen Qual_Fa
The first important featrue for Lasso Regression is: Sale Condition_Alloca
```

In [23]:
```python
# Take argsort on the list containing all coefficients
len(elastic_grid.best_estimator_.steps[1][1].coef_)
```

Out[23]: 318

In [26]:
```python
#Visualize the coefficients of ElasticNet
elastic_clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('elastic', ElasticNet(alpha=0.1))])

elastic = elastic_clf.fit(X_train,y_train)

plt.scatter(np.linspace(1,100,318),elastic_grid.best_estimator_.steps[1][1].coef_,c=np
plt.title('ElasticNet Coefficients')
```

Out[26]: Text(0.5, 1.0, 'ElasticNet Coefficients')




```
In [27]: np.argsort(np.absolute(elastic_grid.best_estimator_.steps[1][1].coef_))

Out[27]: array([274, 277, 196, 112, 111,  12,  65, 177, 210, 200, 224,  70, 287,
                 80, 135,  41, 128, 150,  42, 309, 143, 202,  16, 310, 146, 164,
                106, 134, 159, 231, 169, 229, 140, 244, 201,  60, 291, 307, 300,
                269,  19, 225, 303, 305, 161, 306,  58, 199, 239, 257,  71,  84,
                 95,  79, 182, 105, 279, 281, 118,  73, 119, 227, 267,  47, 156,
                 94, 285, 211, 249,  78,  59, 155, 295, 264, 218, 280, 313, 184,
                259, 136,  38, 167, 174, 284,  64, 110, 157, 314, 265, 294, 137,
                238, 131, 191, 102, 121, 237,  81,  53, 272, 148,  90, 152,  34,
                141, 179, 185, 192,  56, 251, 190, 290, 186,  89, 203, 236, 235,
                147, 219, 183, 103, 289, 153, 233, 292, 149, 166,  32, 133, 261,
                250, 283, 104, 256, 223, 198, 204, 216, 100, 299,  61, 139, 171,
                252, 228, 126,  50, 248, 226, 298, 220, 247, 107, 209, 187,  36,
                232,  39,  49, 214, 258, 125,  68, 271,   0, 154,  57, 194, 240,
                273, 115, 242,  72,   3, 278, 205, 217,  26, 304, 262,  48, 130,
                293, 163, 234, 188,  30, 127, 122, 263,  31,  46,  45,  82, 301,
                282, 276, 270, 124,  96,  11, 266, 213, 158, 170, 316, 260, 160,
                 74, 172,  22,  69, 255, 145,   8, 176,  43, 144, 268,  63,  21,
                 37,  28, 180, 246,  29,  40, 101, 315, 311,  23, 215, 117,  66,
                221, 302,  92, 222, 165, 109,  99,  44, 114, 162,  97, 296, 108,
                123, 297, 168, 230, 129,   4, 189, 173,  62, 317, 120,  10,  51,
                275,  67,   1,  33, 175,  88, 312, 308, 207, 151,  52, 116,  24,
                 20,  25,  13, 286, 288,  91,  18, 254, 132, 197, 138, 113, 253,
                  9, 142, 212,   6,  35,  98, 243,   7,   2,  83, 181,  85, 245,
```

```
                 14, 195,  75,  76,  27,  77, 208,  55,  93,  54, 178,  15, 206,
                 86,  87, 193,  17, 241,   5])
```

```
In [31]: # Top three important categorical features
         print("The first important featrue for ElasticNet is: " + preprocessor.transformers_[
         print("The second important featrue for ElasticNet is: " + preprocessor.transformers_
         print("The third important featrue for ElasticNet is: " + preprocessor.transformers_[
```

```
The first important featrue for ElasticNet is: Sale Condition_Abnorml
The second important featrue for ElasticNet is: Sale Condition_Family
The third important featrue for ElasticNet is: Heating QC_TA
```

Task2

February 20, 2019

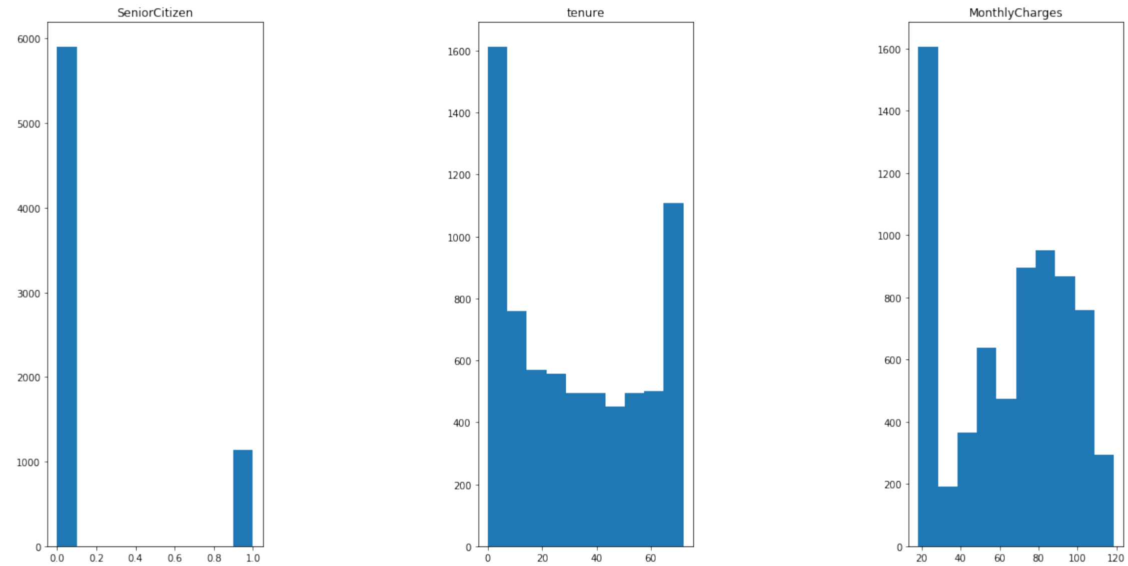
```
In [49]: import warnings
         warnings.filterwarnings('ignore')
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns
         from sklearn.pipeline import make_pipeline
         from sklearn.linear_model import LogisticRegression
         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         from sklearn.pipeline import Pipeline
         from sklearn.preprocessing import OneHotEncoder
         from sklearn.impute import SimpleImputer
         from sklearn.model_selection import cross_val_score
         from sklearn.model_selection import cross_validate
         from sklearn.svm import LinearSVC
         from sklearn.neighbors.nearest_centroid import NearestCentroid
         from sklearn.model_selection import GridSearchCV
         from sklearn_pandas import DataFrameMapper
         from sklearn import preprocessing
         from sklearn.compose import ColumnTransformer
         from sklearn.model_selection import KFold
```

1 2.0: Loading Data

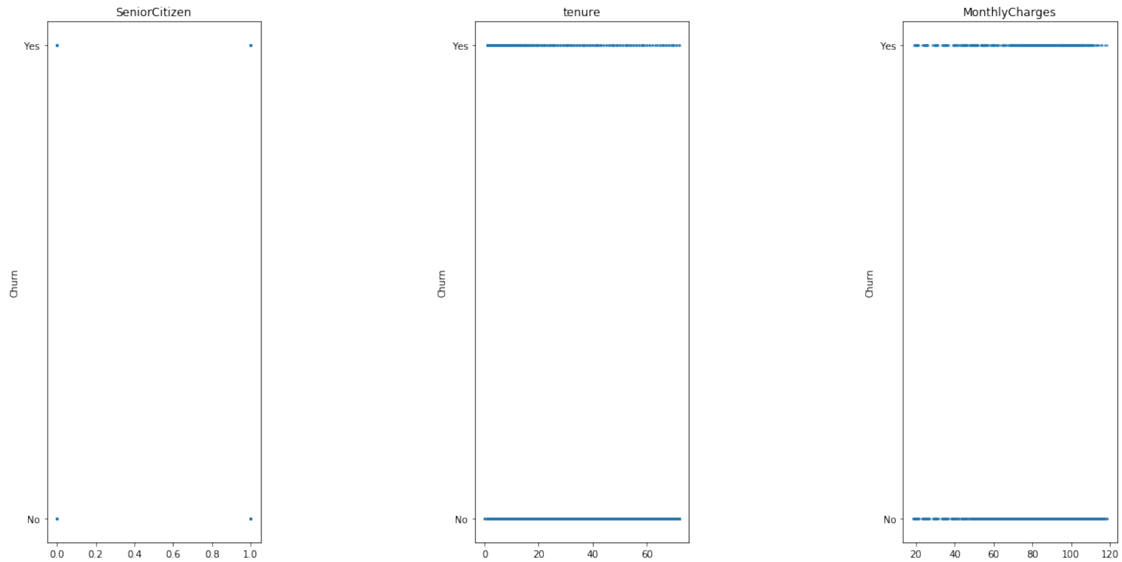
```
In [50]: data = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
         data3 = data.drop(['Churn'], axis=1)
         df = data.select_dtypes(exclude=['object'])
         categ = data.select_dtypes(exclude=['int64','float64'])
         cont_name = list(df)
         categ = categ.drop(['Churn'], axis=1)
         categ_name = list(categ)
         target = data['Churn']
         all_features = data.drop(['Churn'], axis=1)
         df2 = pd.concat([categ, df], axis = 1)
         dep = pd.concat([df, target], axis = 1)
```

2 2.1: Visualize the Univariate Distribution & the Target Feature

In [51]:
```
fig = plt.figure(figsize=(20,10))
fig.subplots_adjust(hspace=0.4, wspace=1)
for i in range(len(df.columns)):
    ax = fig.add_subplot(1, 3, i+1)
    ax.hist(df.iloc[:,i])
    ax.set_title(list(df)[i])
```



In [52]:
```
# Visualize the target feature with each continuous variables
fig = plt.figure(figsize=(20,10))
fig.subplots_adjust(hspace=0.4, wspace=1)
for i in range(df.shape[1]):
    ax = fig.add_subplot(1, 3, i+1)
    ax.scatter(df.iloc[:,i],target,s=1)
    ax.set_title(list(df)[i])
    ax.set_ylabel("Churn")
```



3 2.2: Split Data & Evaluate Models

In [53]:
```
# Create a pipeline with all categorical features for logistic regression
logistic = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore')),
    ('logistic',LogisticRegression())])

# Create a pipeline with all categorical features for SVC
SVC = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore')),
    ('SVC',LinearSVC())])

# Create a pipeline with all categorical features for Nearest Centroid
NC = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(sparse=False,handle_unknown='ignore')),
    ('NC',NearestCentroid())])


# Split data into the training set and the test set
X_train, X_test, y_train, y_test = train_test_split(data3, target, test_size=0.3)

# Fit the logistic regression and evaluate cross validation scores
logistic.fit(X_train, y_train)

scores = cross_val_score(logistic,X_train,y_train, cv=3)
```

```
    scores = sum(scores) / float(len(scores))
    print("Logistic Regression Scores:")
    print(scores)
    print("")


    # Fit the SVC and evaluate cross validation scores
    SVC.fit(X_train, y_train)

    scores = cross_val_score(SVC,X_train,y_train, cv=3)
    scores = sum(scores) / float(len(scores))
    print("SVC Scores:")
    print(scores)
    print("")


    # Fit the Nearest Centroid and evaluate cross validation scores
    NC.fit(X_train, y_train)

    scores = cross_val_score(NC,X_train,y_train, cv=3)
    scores = sum(scores) / float(len(scores))
    print("Nearest Centroid Scores:")
    print(scores)
```

```
Logistic Regression Scores:
0.7886399278514022

SVC Scores:
0.7772816821246123

Nearest Centroid Scores:
0.7002006349037107
```

In [54]:
```
# Create a pipeline with all features and take column trasformation on all features
numeric_transformer = Pipeline(steps=[
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant',fill_value = 'missing')),
    ('onehot', OneHotEncoder(sparse=False, handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num',numeric_transformer, cont_name),
```

```
                ('cat', categorical_transformer, categ_name)])

        # Create a classifier instance for Logistic Regression
        clf1 = Pipeline(steps=[
            ('preprocessor', preprocessor),
            ('logistic', LogisticRegression())])

        # Create a classifier instance for SVC
        clf2 = Pipeline(steps=[
            ('preprocessor', preprocessor),
            ('SVC', LinearSVC())])

        # Create a classifier instance for Nearest Centroid
        clf3 = Pipeline(steps=[
            ('preprocessor', preprocessor),
            ('NC', NearestCentroid())])
```

In [55]:
```
# Split the whole dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(data3, target, test_size=0.3)

# Fit Logistic model and evaluate cross validation scores
clf1.fit(X_train, y_train)
scores = cross_val_score(clf1,X_train,y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("Logistic Regression Scores After Scaling the Continuous features:")
print(scores)
print("")

# Fit SVC model and evaluate cross validation scores
clf2.fit(X_train, y_train)

scores = cross_val_score(clf2,X_train,y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("SVC Scores After Scaling the Continuous features:")
print(scores)
print("")

# Fit Nearest Centroid model and evaluate cross validation scores
clf3.fit(X_train, y_train)

scores = cross_val_score(clf3,X_train,y_train, cv=3)
scores = sum(scores) / float(len(scores))
print("Nearest Centroid Scores After Scaling the Continuous features:")
print(scores)
```

```
Logistic Regression Scores After Scaling the Continuous features:
0.8020261920214798
```

```
SVC Scores After Scaling the Continuous features:
0.8010128984394834

Nearest Centroid Scores After Scaling the Continuous features:
0.7186656926403718
```

4 2.3: Tune the Parameter by the GridSearchCV

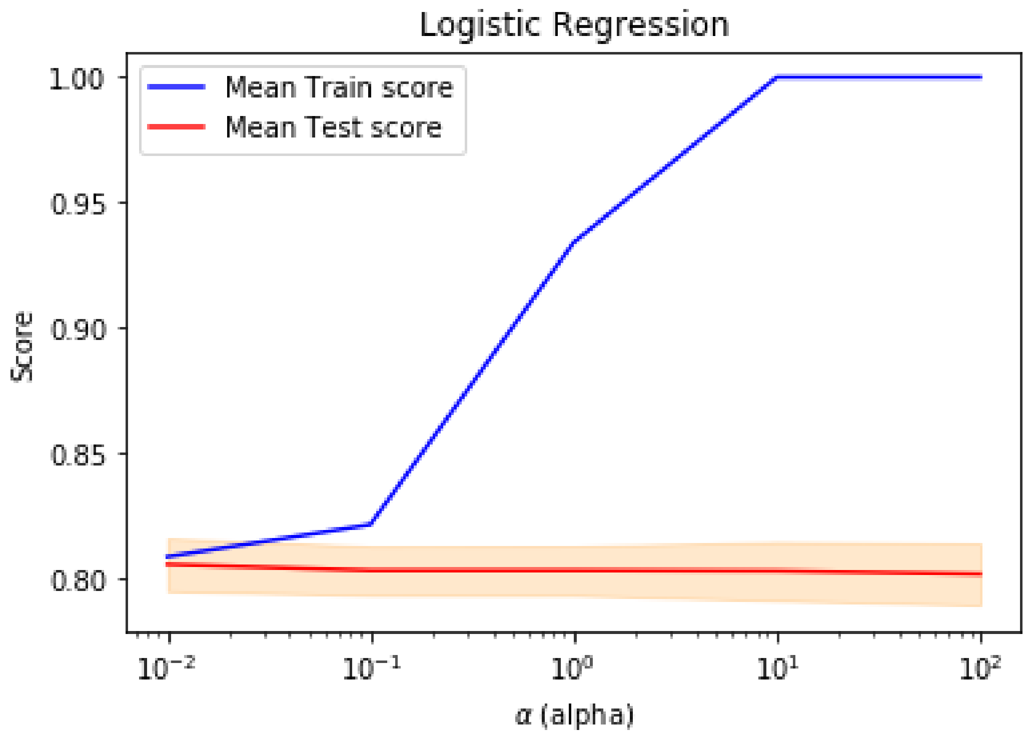
In [56]:
```python
# Search for the best tuning parameter by the GridSearchCV for Logistic Regression
C_param_range = {'logistic__C': [0.01,0.1, 1, 10, 100]}

logistic_grid = GridSearchCV(clf1, param_grid= C_param_range, cv=5)
logistic_grid.fit(X_train, y_train)

print("best mean cross-validation score of Logistic Regression: {:.3f}".format(logisti
print("best parameters of Logistic Regression: {}".format(logistic_grid.best_params_)
print("test-set score of Logistic Regression: {:.3f}".format(logistic_grid.score(X_tes
print("")


# Search for the best tuning parameter by the GridSearchCV for SVC
C_param_range = {'SVC__C': [0.01,0.1, 1, 10, 100]}

SVC_grid = GridSearchCV(clf2, param_grid= C_param_range, cv=5)
SVC_grid.fit(X_train, y_train)

print("best mean cross-validation score of SVC: {:.3f}".format(SVC_grid.best_score_))
print("best parameters of SVC: {}".format(SVC_grid.best_params_))
print("test-set score of SVC: {:.3f}".format(SVC_grid.score(X_test, y_test)))
print("")


# Search for the best tuning parameter by the GridSearchCV for Nearest Centroid
shrink_threshold_param_range = {'NC__shrink_threshold': [0.01,0.1, 1, 10, 100,1000]}

NC_grid = GridSearchCV(clf3, param_grid= shrink_threshold_param_range, cv=5)
NC_grid.fit(X_train, y_train)

print("best mean cross-validation score of SVC: {:.3f}".format(NC_grid.best_score_))
print("best parameters of SVC: {}".format(NC_grid.best_params_))
print("test-set score of SVC: {:.3f}".format(NC_grid.score(X_test, y_test)))
```

```
best mean cross-validation score of Logistic Regression: 0.805
best parameters of Logistic Regression: {'logistic__C': 0.01}
test-set score of Logistic Regression: 0.803
```

```
best mean cross-validation score of SVC: 0.804
best parameters of SVC: {'SVC__C': 0.1}
test-set score of SVC: 0.801

best mean cross-validation score of SVC: 0.740
best parameters of SVC: {'NC__shrink_threshold': 100}
test-set score of SVC: 0.723
```

In [57]:
```python
# Visualize the performance of Logistic Regression
logistic_C = [0.01,0.1, 1, 10, 100]

train_scores_mean = logistic_grid.cv_results_["mean_train_score"]
train_scores_std = logistic_grid.cv_results_["std_train_score"]
test_scores_mean = logistic_grid.cv_results_["mean_test_score"]
test_scores_std = logistic_grid.cv_results_["std_test_score"]

plt.figure()
plt.title('Logistic Regression')
plt.xlabel('$\\alpha$ (alpha)')
plt.ylabel('Score')

plt.semilogx(logistic_C, train_scores_mean, label='Mean Train score',
             color='blue')

plt.semilogx(logistic_C, test_scores_mean,
             label='Mean Test score', color='red')


plt.gca().fill_between(logistic_C,
                       test_scores_mean - test_scores_std,
                       test_scores_mean + test_scores_std,
                       alpha=0.2,
                       color='darkorange')

plt.legend(loc='best')
```

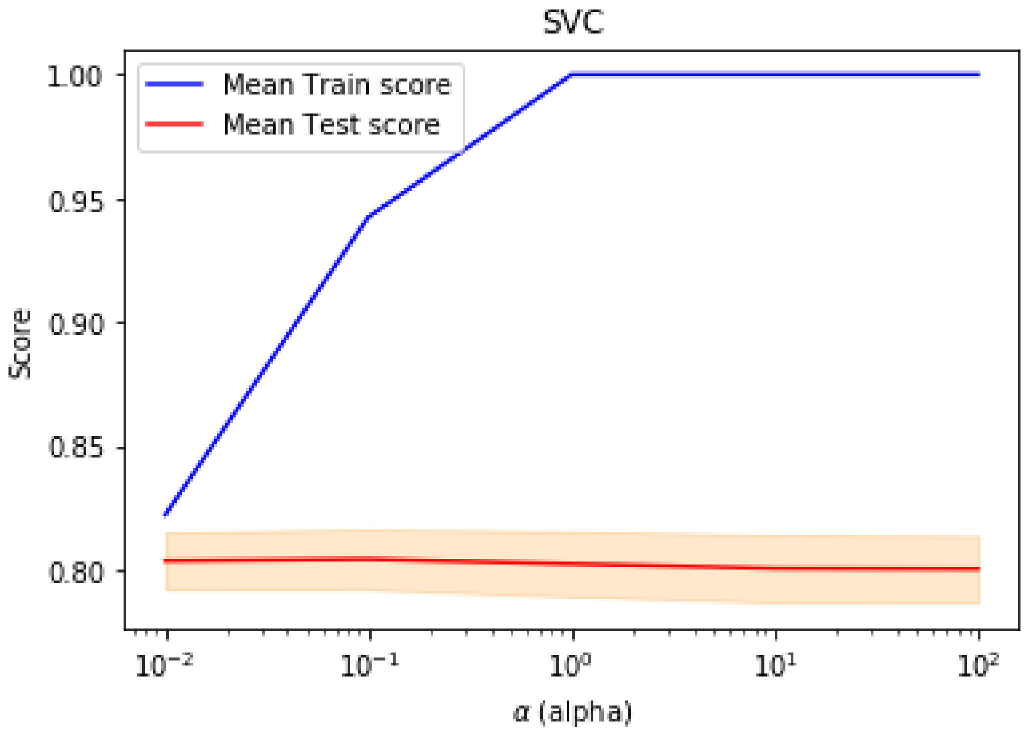
Out[57]:
```
<matplotlib.legend.Legend at 0x1a2bcf3470>
```



In [58]:
```
# Visualize the performance of SVC
SVC_C = [0.01,0.1, 1, 10, 100]

train_scores_mean = SVC_grid.cv_results_["mean_train_score"]
train_scores_std = SVC_grid.cv_results_["std_train_score"]
test_scores_mean = SVC_grid.cv_results_["mean_test_score"]
test_scores_std = SVC_grid.cv_results_["std_test_score"]

plt.figure()
plt.title('SVC')
plt.xlabel('$\\alpha$ (alpha)')
plt.ylabel('Score')

plt.semilogx(SVC_C, train_scores_mean, label='Mean Train score',
             color='blue')

plt.semilogx(SVC_C, test_scores_mean,
             label='Mean Test score', color='red')


plt.gca().fill_between(SVC_C,
                       test_scores_mean - test_scores_std,
                       test_scores_mean + test_scores_std,
```

```
                          alpha=0.2,
                          color='darkorange')

plt.legend(loc='best')
```

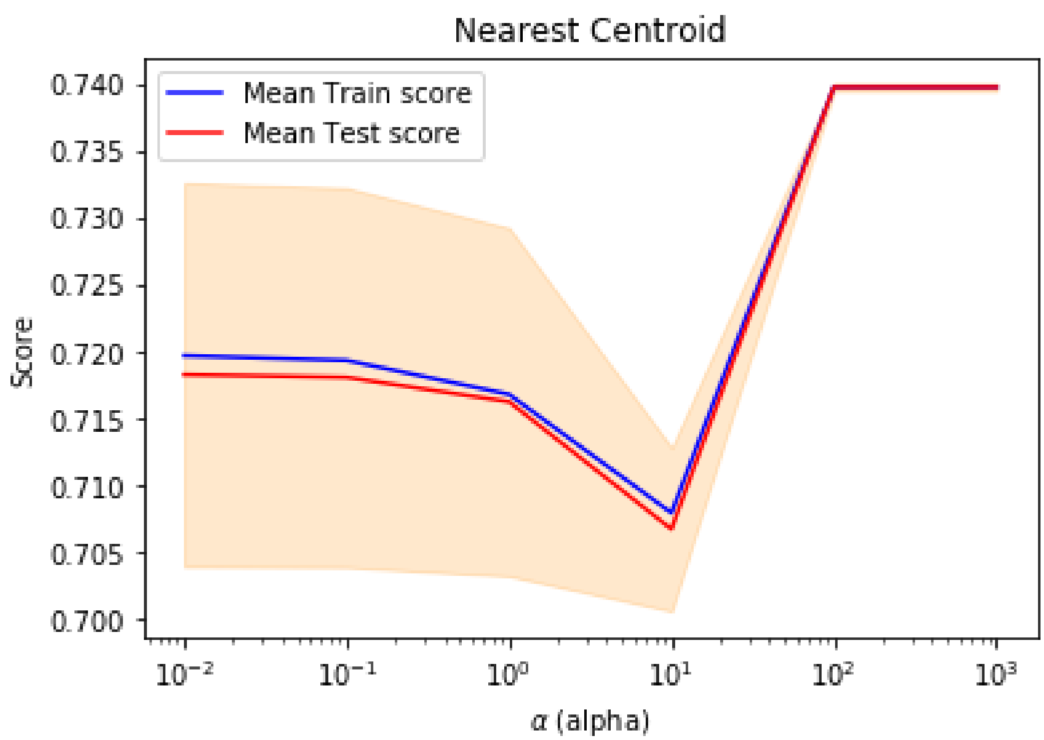
Out[58]: <matplotlib.legend.Legend at 0x1a32c92ac8>



In [59]:
```
# Visualize the performance of Nearest Centroid
NC_C = [0.01,0.1, 1, 10, 100,1000]

train_scores_mean = NC_grid.cv_results_["mean_train_score"]
train_scores_std = NC_grid.cv_results_["std_train_score"]
test_scores_mean = NC_grid.cv_results_["mean_test_score"]
test_scores_std = NC_grid.cv_results_["std_test_score"]

plt.figure()
plt.title('Nearest Centroid')
plt.xlabel('$\\alpha$ (alpha)')
plt.ylabel('Score')

plt.semilogx(NC_C, train_scores_mean, label='Mean Train score',
             color='blue')
```

```
plt.semilogx(NC_C, test_scores_mean,
             label='Mean Test score', color='red')


plt.gca().fill_between(NC_C,
                       test_scores_mean - test_scores_std,
                       test_scores_mean + test_scores_std,
                       alpha=0.2,
                       color='darkorange')

plt.legend(loc='best')
```

Out[59]: <matplotlib.legend.Legend at 0x1a2c00eb00>



5 Conclusion on 2.3

Overall, the mean train score get better and better for Rogistic Regression and SVC as the alpha increases. However, the mean test scores for both pretty much stagnate regardless of the alpha. For Nearest Centroid, the mean train score and mean test score behave almost the same. An interesting pattern is that apparently as the alpha increases, both scores increase as well. However, an alpha bigger than 10^2 no longer improves the scores.

6 2.4: Change the Cross-Validation Strategy

In [60]:
```
# Search for the best tuning parameter by the GridSearchCV with Shuffle for Logistic 
C_param_range = {'logistic__C': [0.01,0.1, 1, 10, 100]}
k_fold = KFold(n_splits=5,random_state=None, shuffle=True)
logistic_grid = GridSearchCV(clf1, param_grid= C_param_range, cv=k_fold)
logistic_grid.fit(data3.iloc[train], target.iloc[train])



print("best mean cross-validation score of Logistic Regression with shuffle: {:.3f}".
print("best parameters of Logistic Regression with shuffle: {}".format(logistic_grid.
print("test-set score of Logistic Regression with shuffle: {:.3f}".format(logistic_gr
print("")

# Search for the best tuning parameter by the GridSearchCV with Shuffle for SVC
C_param_range = {'SVC__C': [0.01,0.1, 1, 10, 100]}
k_fold = KFold(n_splits=2,random_state=None, shuffle=True)
SVC_grid = GridSearchCV(clf2, param_grid= C_param_range, cv=k_fold)
SVC_grid.fit(data3.iloc[train], target.iloc[train])

print("best mean cross-validation score of SVC with shuffle: {:.3f}".format(SVC_grid.
print("best parameters of SVC with shuffle: {}".format(SVC_grid.best_params_))
print("test-set score of SVC with shuffle: {:.3f}".format(SVC_grid.score(X_test, y_te
print("")


# Search for the best tuning parameter by the GridSearchCV with Shuffle for Nearest C
shrink_threshold_param_range = {'NC__shrink_threshold': [0.01,0.1, 1, 10, 100]}
k_fold = KFold(n_splits=2,random_state=None, shuffle=True)
NC_grid = GridSearchCV(clf3, param_grid= shrink_threshold_param_range, cv=k_fold)
NC_grid.fit(data3.iloc[train], target.iloc[train])

print("best mean cross-validation score of Nearest Centroid with shuffle: {:.3f}".for
print("best parameters of Nearest Centroid with shuffle: {}".format(NC_grid.best_para
print("test-set score of Nearest Centroid with shuffle: {:.3f}".format(NC_grid.score(
```

```
best mean cross-validation score of Logistic Regression with shuffle: 0.804
best parameters of Logistic Regression with shuffle: {'logistic__C': 0.01}
test-set score of Logistic Regression with shuffle: 0.804

best mean cross-validation score of SVC with shuffle: 0.806
best parameters of SVC with shuffle: {'SVC__C': 0.01}
test-set score of SVC with shuffle: 0.818

best mean cross-validation score of Nearest Centroid with shuffle: 0.737
best parameters of Nearest Centroid with shuffle: {'NC__shrink_threshold': 100}
test-set score of Nearest Centroid with shuffle: 0.723
```

In [61]:
```
# Search for the best tuning parameter by the GridSearchCV with Shuffle and Random St
C_param_range = {'logistic__C': [0.01,0.1, 1, 10, 100]}
k_fold = KFold(n_splits=2,random_state=123, shuffle=True)
logistic_grid = GridSearchCV(clf1, param_grid= C_param_range, cv=k_fold)
logistic_grid.fit(data3.iloc[train], target.iloc[train])

print("best mean cross-validation score of Logistic Regression with shuffle: {:.3f}".
print("best parameters of Logistic Regression with shuffle: {}".format(logistic_grid.
print("test-set score of Logistic Regression with shuffle: {:.3f}".format(logistic_gr
print("")


# Search for the best tuning parameter by the GridSearchCV with Shuffle and Random St
C_param_range = {'SVC__C': [0.01,0.1, 1, 10, 100]}
k_fold = KFold(n_splits=2,random_state=None, shuffle=True)
SVC_grid = GridSearchCV(clf2, param_grid= C_param_range, cv=k_fold)
SVC_grid.fit(data3.iloc[train], target.iloc[train])

print("best mean cross-validation score of SVC with shuffle: {:.3f}".format(SVC_grid.
print("best parameters of SVC with shuffle: {}".format(SVC_grid.best_params_))
print("test-set score of SVC with shuffle: {:.3f}".format(SVC_grid.score(X_test, y_te
print("")

# Search for the best tuning parameter by the GridSearchCV with Shuffle and Random St
shrink_threshold_param_range = {'NC__shrink_threshold': [0.01,0.1, 1, 10, 100]}
k_fold = KFold(n_splits=2,random_state=None, shuffle=True)
NC_grid = GridSearchCV(clf3, param_grid= shrink_threshold_param_range, cv=k_fold)
NC_grid.fit(data3.iloc[train], target.iloc[train])

print("best mean cross-validation score of Nearest Centroid with shuffle: {:.3f}".for
print("best parameters of Nearest Centroid with shuffle: {}".format(NC_grid.best_para
print("test-set score of Nearest Centroid with shuffle: {:.3f}".format(NC_grid.score(
```

```
best mean cross-validation score of Logistic Regression with shuffle: 0.802
best parameters of Logistic Regression with shuffle: {'logistic__C': 0.1}
test-set score of Logistic Regression with shuffle: 0.815

best mean cross-validation score of SVC with shuffle: 0.805
best parameters of SVC with shuffle: {'SVC__C': 0.1}
test-set score of SVC with shuffle: 0.912

best mean cross-validation score of Nearest Centroid with shuffle: 0.737
best parameters of Nearest Centroid with shuffle: {'NC__shrink_threshold': 100}
test-set score of Nearest Centroid with shuffle: 0.723
```

7 2.5: Visualize the Coefficients

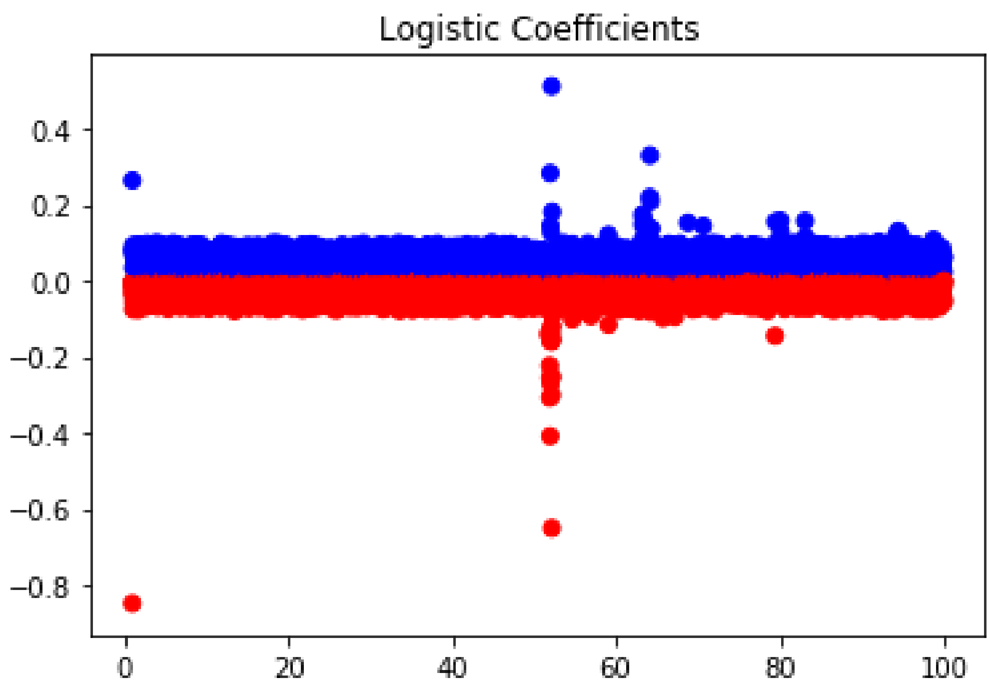
In [62]:
```
# Instantiate an instance for Logistic Regression and SVC with the best parameter obt
clf1 = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('logistic', LogisticRegression(C=1))])

clf2 = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('SVC', LinearSVC(C=0.1))])
```

In [65]:
```
#Sha
logistic_grid.best_estimator_.steps[1][1].coef_.shape
```

Out[65]: (1, 10970)

In [66]:
```
# Plot coefficients of Logistic Regression
logistic = clf1.fit(X_train,y_train)
plt.scatter(np.linspace(1,100,10970),logistic_grid.best_estimator_.steps[1][1].coef_,c
plt.title('Logistic Coefficients')
```

Out[66]: Text(0.5, 1.0, 'Logistic Coefficients')



In [213]:
```
# Plot coefficients of SVC
SVC = clf2.fit(X_train,y_train)
plt.scatter(np.linspace(1,100,6924),SVC_grid.best_estimator_.steps[1][1].coef_,c=np.s
plt.title('SVC Coefficients')
```

```
Out[213]: Text(0.5, 1.0, 'SVC Coefficients')
```