Homework 2 - Task 1

Michelle Chen (mc4571)

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [195]:
```
cali = pd.read_excel("AmesHousing.xls")
cali = cali.drop(columns = ['Order', 'PID'])
cali_t = cali.copy()
y = cali['SalePrice']
cali = cali.iloc[:,:-1]
cali.head()
```

Out[195]:

	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	Lot Config	...	Scre Por
0	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	AllPub	Corner	...	
1	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Inside	...	1
2	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corner	...	
3	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	AllPub	Corner	...	
4	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Inside	...	

5 rows × 79 columns

In [39]:
```
cali.shape
```

Out[39]: (2930, 79)

Task 1

Part 1.1

In [40]:
```
cont_col = cali._get_numeric_data().columns
cont_col = list(cont_col)
```

In [41]:
```
cont_col = [e for e in cont_col if e not in ('MS SubClass', 'Overall Qual', 'O
verall Cond', 'SalePrice')]
cont_col, len(cont_col)
```

Out[41]:
```
(['Lot Frontage',
  'Lot Area',
  'Year Built',
  'Year Remod/Add',
  'Mas Vnr Area',
  'BsmtFin SF 1',
  'BsmtFin SF 2',
  'Bsmt Unf SF',
  'Total Bsmt SF',
  '1st Flr SF',
  '2nd Flr SF',
  'Low Qual Fin SF',
  'Gr Liv Area',
  'Bsmt Full Bath',
  'Bsmt Half Bath',
  'Full Bath',
  'Half Bath',
  'Bedroom AbvGr',
  'Kitchen AbvGr',
  'TotRms AbvGrd',
  'Fireplaces',
  'Garage Yr Blt',
  'Garage Cars',
  'Garage Area',
  'Wood Deck SF',
  'Open Porch SF',
  'Enclosed Porch',
  '3Ssn Porch',
  'Screen Porch',
  'Pool Area',
  'Misc Val',
  'Mo Sold',
  'Yr Sold'],
 33)
```

In [42]:
```
cat_col = list(set(cali.columns) - set(cont_col))
cat_col
```

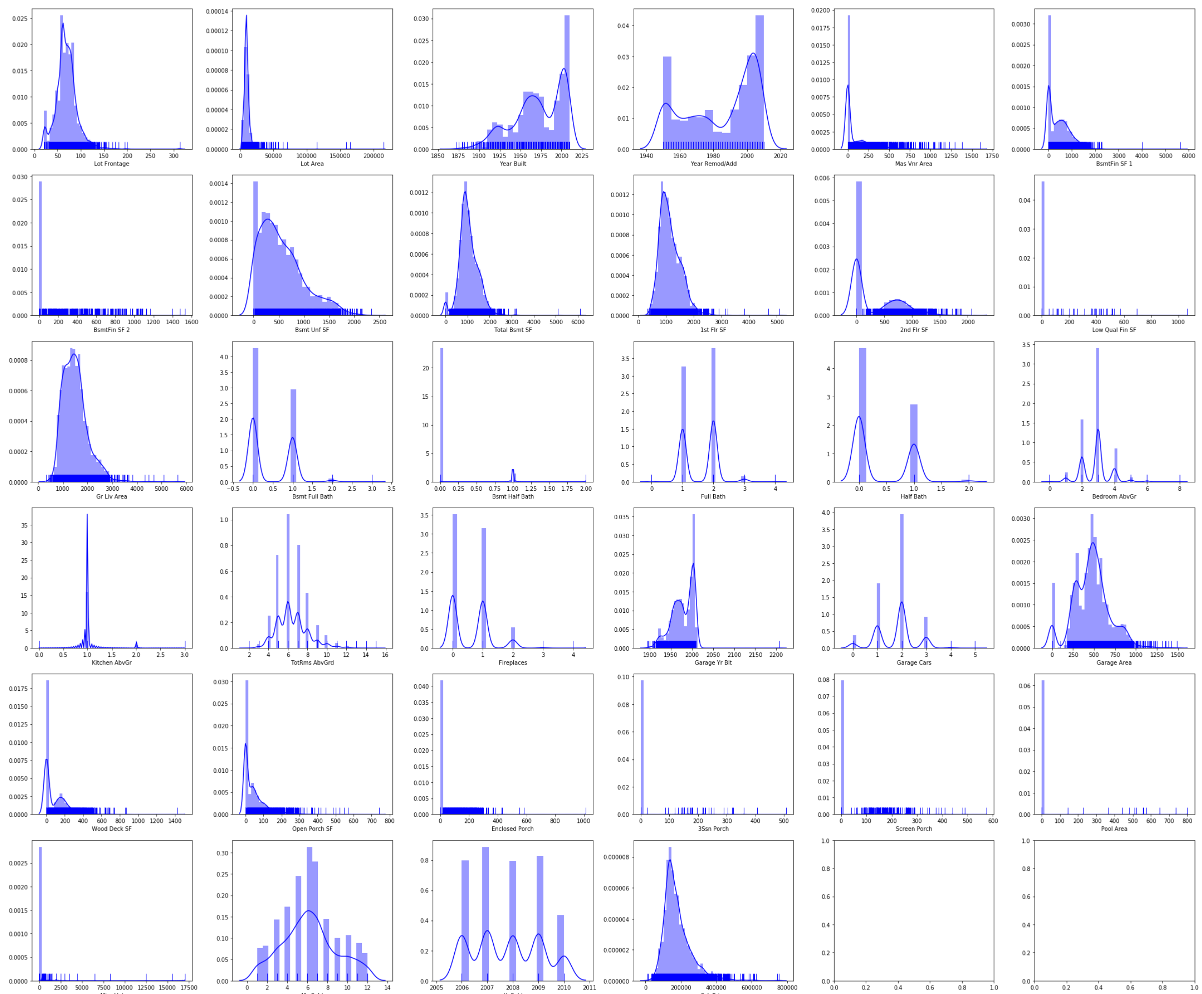
Out[42]:
```
['Garage Qual',
 'Heating',
 'Foundation',
 'Exterior 1st',
 'Bldg Type',
 'Garage Finish',
 'Neighborhood',
 'Mas Vnr Type',
 'Central Air',
 'Fireplace Qu',
 'MS Zoning',
 'Alley',
 'Misc Feature',
 'Paved Drive',
 'Lot Shape',
 'Overall Cond',
 'Land Contour',
 'Exter Cond',
 'Bsmt Cond',
 'House Style',
 'BsmtFin Type 1',
 'BsmtFin Type 2',
 'Heating QC',
 'Sale Type',
 'MS SubClass',
 'Exterior 2nd',
 'Bsmt Qual',
 'Functional',
 'Bsmt Exposure',
 'Electrical',
 'Fence',
 'Land Slope',
 'Garage Cond',
 'Sale Condition',
 'Exter Qual',
 'Roof Matl',
 'Lot Config',
 'Roof Style',
 'Kitchen Qual',
 'Utilities',
 'Condition 2',
 'Overall Qual',
 'Garage Type',
 'Condition 1',
 'Pool QC',
 'Street']
```

In [43]:
```
cont_col_target = cont_col.copy()
cont_col_target.append('SalePrice')
```

In [44]:
```
cont_col_target
```

Out[44]:
```
['Lot Frontage',
 'Lot Area',
 'Year Built',
 'Year Remod/Add',
 'Mas Vnr Area',
 'BsmtFin SF 1',
 'BsmtFin SF 2',
 'Bsmt Unf SF',
 'Total Bsmt SF',
 '1st Flr SF',
 '2nd Flr SF',
 'Low Qual Fin SF',
 'Gr Liv Area',
 'Bsmt Full Bath',
 'Bsmt Half Bath',
 'Full Bath',
 'Half Bath',
 'Bedroom AbvGr',
 'Kitchen AbvGr',
 'TotRms AbvGrd',
 'Fireplaces',
 'Garage Yr Blt',
 'Garage Cars',
 'Garage Area',
 'Wood Deck SF',
 'Open Porch SF',
 'Enclosed Porch',
 '3Ssn Porch',
 'Screen Porch',
 'Pool Area',
 'Misc Val',
 'Mo Sold',
 'Yr Sold',
 'SalePrice']
```

In [48]:
```
fig, ax = plt.subplots(6,6,figsize=(30, 25))

k = 0
for i in range(0,6):
    for j in range(0,6):
        if k < 34:
            sns.distplot(cali_t[cont_col_target[k]][~np.isnan(cali_t[cont_col_
target[k]])], color = 'blue', rug = True, ax = ax[i,j])
            k = k+1


plt.tight_layout()
```

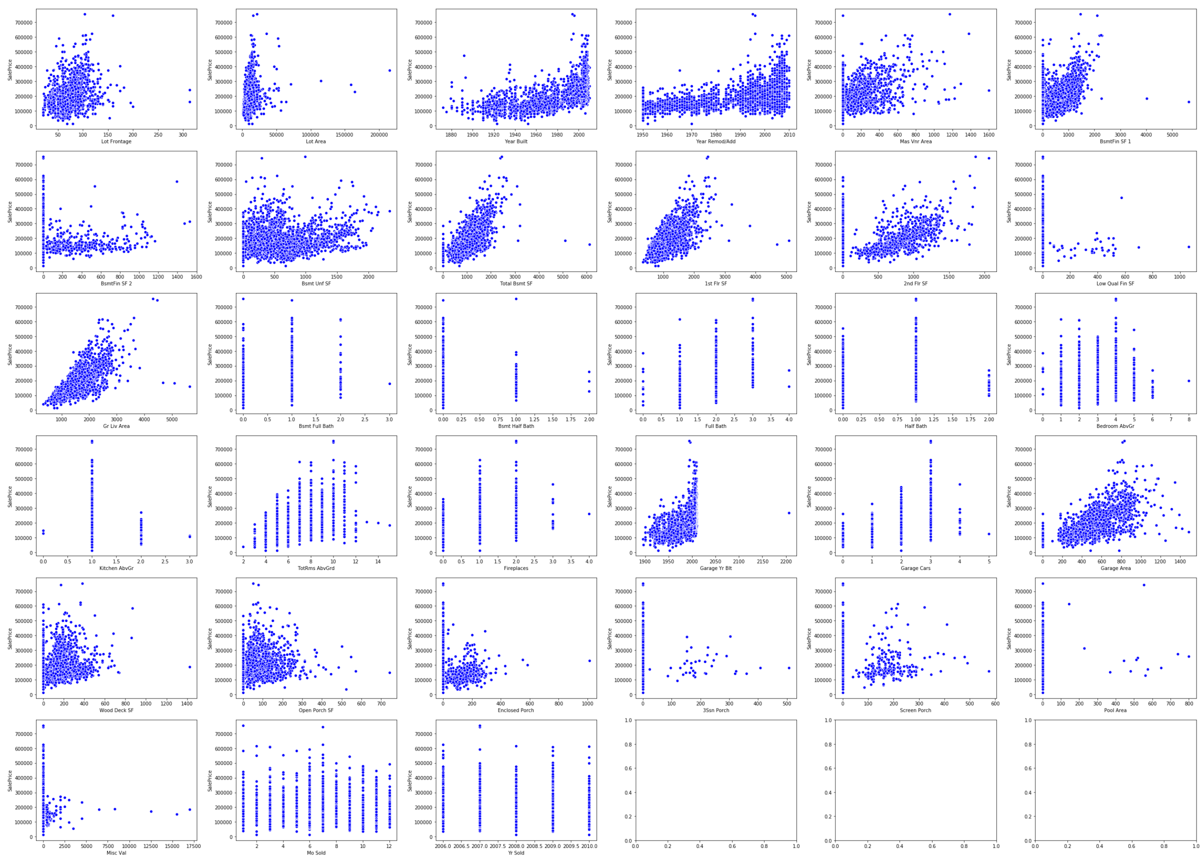


Insights:

When reading in the data we see that many of the variables can be considered categorical or continuous. I originally partitioned the variables using their respective dtype. I soon realized that certain variables such as month/year variables could be either (but I chose to categorize them as continuous variables). Other variables such as MS Subclass, while numeric, represented types of dwellings and therefore should be categorical. I also treated discrete numerical variables (ie. # of bathrooms) as a continuous variables in this assignment. Other observations about the data upon plotting it include that the distributions are all typically left skewed, right-tailed or are multimodal (for the year/month variables). The former implies that most of the values within the distributions occur to the lower end of the spectrum, whereas the latter means that there are certain values that occur more frequently.

1.2: Visualize the dependency of the target on each continuous feature (2d scatter plot)

In [46]:
```
fig, ax = plt.subplots(6,6,figsize=(35, 25))

k = 0
for i in range(0,6):
    for j in range(0,6):
        if k < 33:
            sns.scatterplot(cali[cont_col[k]][~np.isnan(cali[cont_col[k]])], y
, color = 'b', ax = ax[i,j])
            k+=1

plt.tight_layout()
```



1.3: Split data in training and test set

In [49]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(cali, y, random_state = 0)
```

In [50]:
```
len(X_train) #75%
```

Out[50]: 2197

In [51]:
```
len(X_test) #25%
```
Out[51]: 733

In [52]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
```

In [53]:
```
cat_col = X_train.dtypes == object
len(cat_col[cat_col==True])
cat_col = list(cat_col[cat_col == True].index)
len(cat_col)
```
Out[53]: 43

In [55]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score

X_train.loc[:,cat_col] = X_train.loc[:,cat_col].fillna('NaN')

r2_matrix = []

for i in cat_col:
    ce = OneHotEncoder(handle_unknown='ignore').fit_transform(X_train[i].astyp
e(str).values.reshape(-1,1))
    r2_score = cross_val_score(LinearRegression(), ce, pd.DataFrame(y_train),
cv=5, scoring = 'r2')
    print(i, np.mean(r2_score))
    r2_matrix.append(np.mean(r2_score))
```

```
MS Zoning 0.10706333110216078
Street -0.0005413468847056402
Alley 0.018212727100749837
Lot Shape 0.08805295079080108
Land Contour 0.0419931262854605
Utilities -0.002648992833832864
Lot Config 0.010237767581507119
Land Slope -0.0001641626415472608
Neighborhood 0.5612594768619081
Condition 1 0.028591050211333124
Condition 2 0.008909560147208118
Bldg Type 0.03275234243932841
House Style 0.0653148161837211
Roof Style 0.061596034878051764
Roof Matl 0.003553750904714903
Exterior 1st 0.1590607523289052
Exterior 2nd 0.15031033286163165
Mas Vnr Type 0.2073201890731084
Exter Qual 0.5217375382947376
Exter Cond 0.019668868338641808
Foundation 0.27068629605799355
Bsmt Qual 0.5107566408258944
Bsmt Cond 0.04317715716870498
Bsmt Exposure 0.1800097014751591
BsmtFin Type 1 0.22422320619972816
BsmtFin Type 2 0.025553954205057218
Heating 0.002687812778743748
Heating QC 0.21265665364659073
Central Air 0.06845855900548314
Electrical 0.05270790125550144
Kitchen Qual 0.4858650215487065
Functional 0.013248075401956317
Fireplace Qu 0.3198799861031086
Garage Type 0.23502536437616958
Garage Finish 0.300336164298868
Garage Qual 0.08486929671555118
Garage Cond 0.07899710520371113
Paved Drive 0.0742086167306042
Pool QC -0.004421329762172443
Fence 0.03950161046878779
Misc Feature -0.0036980943016864166
Sale Type 0.1234076526977023
Sale Condition 0.12383528526632566
```

In [56]:

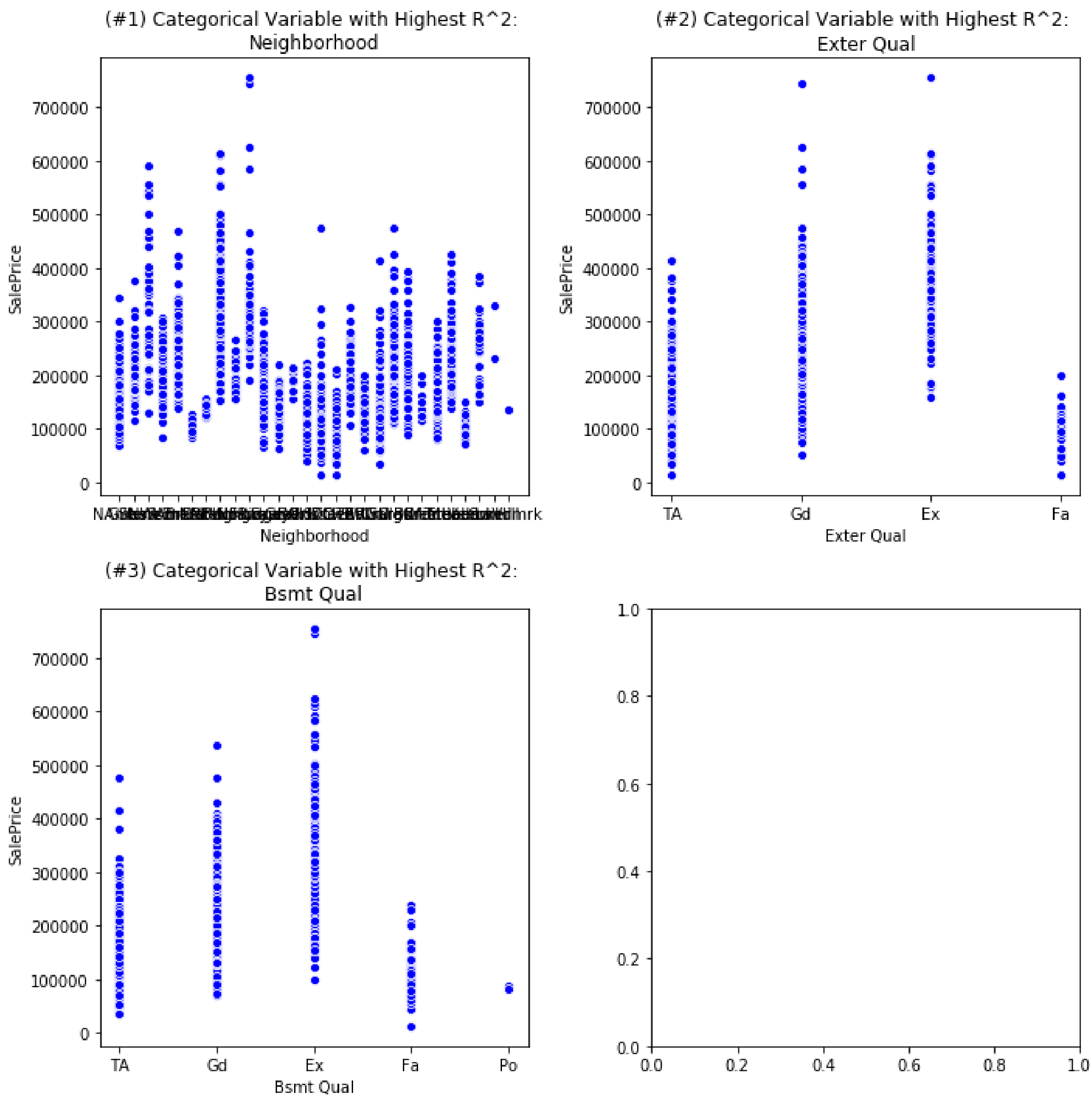
```
r2_matrix = pd.DataFrame(r2_matrix)
r2_sorted = r2_matrix.sort_values(by = [0], ascending = False).head(3)
r2_sorted
```

Out[56]:

	0
8	0.561259
18	0.521738
21	0.510757

In [71]:
```
first = 8
second = 18
third = 21


f = cat_col[first]
s = cat_col[second]
t = cat_col[third]

f,s,t

r2_columns = [f, s, t]
```

In [84]:
```
# Plotting Variables with highest r^2

fig, ax = plt.subplots(2,2,figsize=(10,10))
k = 0

for i in range(0,2):
    for j in range(0,2):
        if k < 3:
            sns.scatterplot(cali[r2_columns[k]], y, color = 'blue', ax = ax[i,
j], x_jitter = True)
            ax[i,j].set_title('(#' + str(k+1) +  ') Categorical Variable with
 Highest R^2: \n' +  r2_columns[k])
            k +=1

plt.tight_layout()
```



1.4 Use column transformer

In [90]:
```
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
```

In [91]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(cali, y, random_state = 0)
```

In [92]:
```
X_train.loc[:,cat_col] = X_train.loc[:,cat_col].fillna('NaN')
```

Linear Regression

In [93]:
```
# Nonstandardized
preprocess_nonscale = make_column_transformer((SimpleImputer(strategy = 'media
n'), cont_col),
                                              (OneHotEncoder(handle_unknown='i
gnore'), cat_col))
model = make_pipeline(preprocess_nonscale,LinearRegression())
scores_lr_nonscale = cross_val_score(model, X_train, y_train, cv=5, scoring =
'r2')

# Standardized
preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'),LinearReg
ression())
scores_lr = cross_val_score(model, X_train, y_train, cv=5, scoring = 'r2')
```

In [94]:
```
np.mean(scores_lr_nonscale), np.mean(scores_lr)
```

Out[94]:
```
(0.8318130959054899, 0.8224394325074629)
```

Insights:

Scaling the data within the pipeline using StandardScaler does not actually improve our final cross validation score. In fact scaling the continuous data actually worsened our model's cross validated r^2 score.

Ridge Regression

In [111]:
```
# Nonstandardized
from sklearn.linear_model import Ridge
preprocess_nonscale = make_column_transformer((SimpleImputer(strategy = 'media
n'), cont_col),
                                              (OneHotEncoder(handle_unknown='i
gnore'), cat_col))
model = make_pipeline(preprocess_nonscale, Ridge())
scores_rr_nonscale = cross_val_score(model, X_train, y_train, cv=5, scoring =
'r2')

# Standardized
preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'),Ridge())
scores_rr = cross_val_score(model, X_train, y_train, cv=5, scoring = 'r2')
```

In [112]:
```
np.mean(scores_rr_nonscale), np.mean(scores_rr)
```

Out[112]:
```
(0.6964827854333498, 0.8600458483029151)
```

Insights:

Scaling the data within the pipeline using StandardScaler drastically improved our final mean cross validation score!

Lasso Regression

In [96]:
```
from sklearn.linear_model import Lasso

# Nonstandardized
preprocess_nonscale = make_column_transformer((SimpleImputer(strategy = 'media
n'), cont_col),
                                              (OneHotEncoder(handle_unknown='i
gnore'), cat_col))
model = make_pipeline(preprocess_nonscale, Lasso())
scores_lasso_nonscale = cross_val_score(model, X_train, y_train, cv=5, scoring
= 'r2')

# Standardized
preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'),Lasso())
scores_lasso = cross_val_score(model, X_train, y_train, cv=5, scoring = 'r2')
```

In [97]:
```
np.mean(scores_lasso_nonscale), np.mean(scores_lasso)
```

Out[97]:
```
(0.8578447550840751, 0.8578923298866318)
```

Insights:

Scaling the data within the pipeline using StandardScaler did not significantly improve our mean cross validated r^2 score. In the case with Lasso regression, standardscaler does not affect the outcome as much as it did with Ridge Regression.

Elastic Net

In [98]:
```
#Elastic Net
from sklearn.linear_model import ElasticNet

# Nonstandardized
preprocess_nonscale = make_column_transformer((SimpleImputer(strategy = 'media
n'), cont_col),
                                              (OneHotEncoder(handle_unknown='i
gnore'), cat_col))
model = make_pipeline(preprocess_nonscale, ElasticNet())
scores_en_nonscale = cross_val_score(model, X_train, y_train, cv=5, scoring =
'r2')

# Standardized
preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), ElasticN
et())
scores_en = cross_val_score(model, X_train, y_train, cv=5, scoring = 'r2')
```

In [99]:
```
np.mean(scores_en_nonscale), np.mean(scores_en)
```

Out[99]:
```
(0.8163294726002126, 0.8307526116245411)
```

Insights:

Scaling the data within the pipeline using StandardScaler did help boost our final mean cross validated r^2 score slightly, but the effects were not as drastic as with ridge regression.

1.5: Tune the parameters of models using GridSearchCV

Linear Regression Tuning

In [101]:
```
from sklearn.model_selection import GridSearchCV
```

In [58]:
```
param_grid = {'linearregression__fit_intercept':('True', 'False'), 'linearregr
ession__normalize':('True', 'False')}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), LinearRe
gression())

grid = GridSearchCV(model, param_grid=param_grid,
                    cv=5, return_train_score=True, scoring = 'r2')

grid = grid.fit(X_train, y_train)

print("Linear Regression - best mean cross-validation score: {:.3f}".format(gr
id.best_score_))
print("Linear Regression - best parameters: {}".format(grid.best_params_))
```

```
Linear Regression - best mean cross-validation score: 0.841
Linear Regression - best parameters: {'linearregression__fit_intercept': 'Tru
e', 'linearregression__normalize': 'True'}
```

Insights:

Tuning the basic parameters for LR using GridSearchCV did improve our mean cross validation score slightly from 0.8224 -> 0.841.

Ridge Regression tuning

In [113]:
```
# Ridge Regression
param_grid2 = {'ridge__alpha': np.logspace(-3, 3, 13)}
preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), Ridge())

grid2 = GridSearchCV(model, param_grid=param_grid2,
                    cv=5, return_train_score=True, scoring = 'r2')

grid2.fit(X_train, y_train)

print("RR - best mean cross-validation score: {:.3f}".format(grid2.best_score_
))
print("RR - best parameters: {}".format(grid2.best_params_))
```

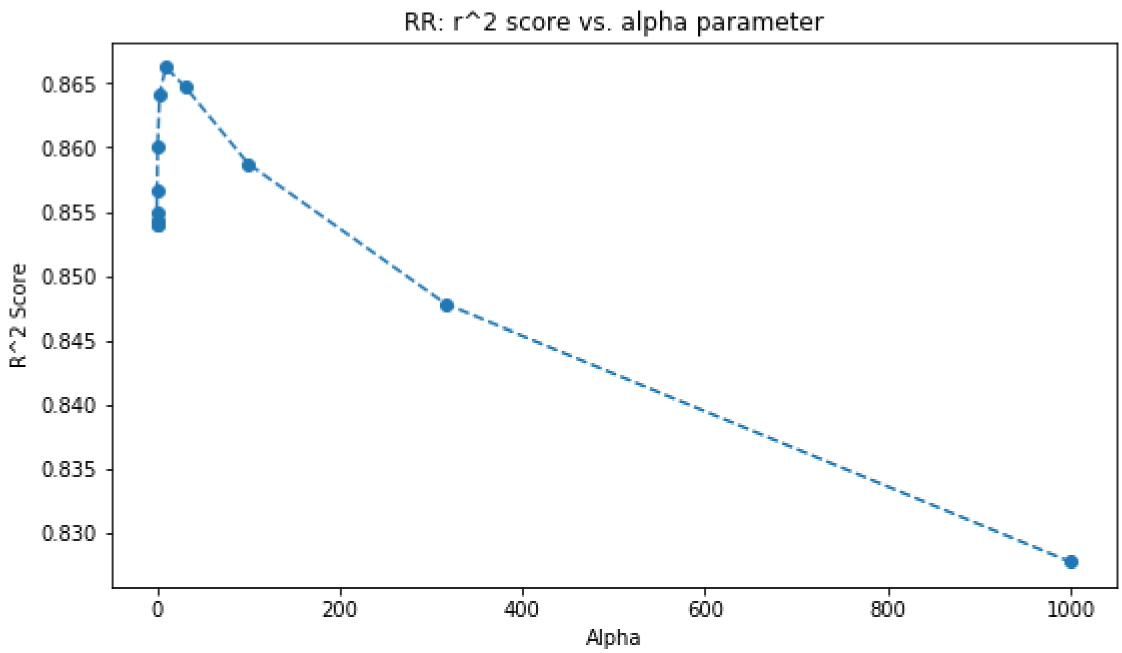
```
RR - best mean cross-validation score: 0.866
RR - best parameters: {'ridge__alpha': 10.0}
```

Insights:

Iterating over the possible alpha parameters during GridSearchCV enabled us to improve our mean cross validation score slightly for RR from 0.860 -> 0.866.

In [114]:
```
x_score = pd.DataFrame(grid2.cv_results_['mean_test_score'])
parameters = pd.DataFrame(param_grid2['ridge__alpha'])

fig = plt.figure(figsize=(9, 5))
ax = plt.gca()

ax.plot(parameters, x_score, marker='o', linestyle='dashed')
ax.set_title('RR: r^2 score vs. alpha parameter')
ax.set_xlabel('Alpha')
ax.set_ylabel('R^2 Score')
```

Out[114]:
```
Text(0,0.5,'R^2 Score')
```



Lasso regression tuning

In [103]:
```
# Lasso Regression

param_grid3 = {'lasso__alpha': np.logspace(-3, 0, 13)}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), Lasso())

grid3 = GridSearchCV(model, param_grid=param_grid3,
                    cv=5, return_train_score=True, scoring = 'r2')

grid3.fit(X_train, y_train)
print("Lasso Regression - best mean cross-validation score: {:.3f}".format(gri
d3.best_score_))
print("Lasso Regression - best parameters: {}".format(grid3.best_params_))
```

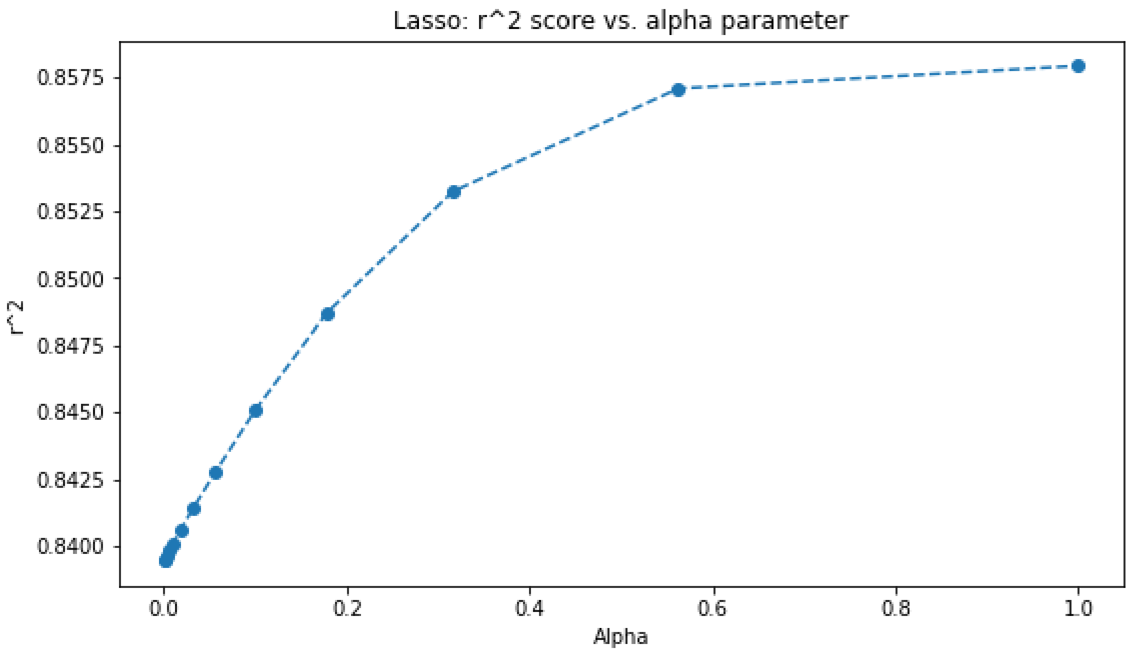
```
Lasso Regression - best mean cross-validation score: 0.858
Lasso Regression - best parameters: {'lasso__alpha': 1.0}
```

Insights:

Iterating over the possible alpha parameters during GridSearchCV enabled us to improve our mean cross validation score slightly for Lasso Regression from 0.8578 -> 0.858.

In [106]:
```
x_score = pd.DataFrame(grid3.cv_results_['mean_test_score'])
parameters = pd.DataFrame(param_grid3['lasso__alpha'])

fig = plt.figure(figsize=(9, 5))
ax = plt.gca()

ax.plot(parameters, x_score, marker='o', linestyle='dashed')
ax.set_title('Lasso: r^2 score vs. alpha parameter')
ax.set_xlabel('Alpha')
ax.set_ylabel('r^2')
```

Out[106]:
```
Text(0,0.5,'r^2')
```



Elastic Net Tuning

In [105]:
```
param_grid4 = {'elasticnet__alpha': np.logspace(-4, -1, 10),
               'elasticnet__l1_ratio': [0.01, .1, .5, .9, .98, 1]}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), ElasticN
et())

grid4 = GridSearchCV(model, param_grid=param_grid4,
                    cv=5, return_train_score=True, scoring = 'r2')

grid4.fit(X_train, y_train)

print("Elastic Net Regression - best mean cross-validation score: {:.3f}".form
at(grid4.best_score_))
print("Elastic Net Regression - best parameters: {}".format(grid4.best_params_
))
```

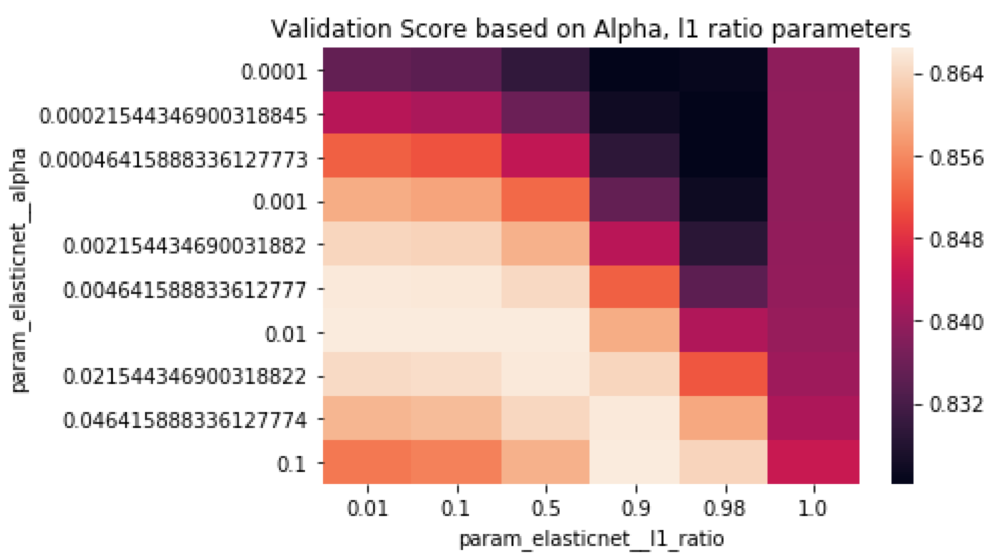
```
Elastic Net Regression - best mean cross-validation score: 0.866
Elastic Net Regression - best parameters: {'elasticnet__alpha': 0.01, 'elasti
cnet__l1_ratio': 0.1}
```

Insights:

Iterating over the possible alpha and l1_ratio parameters during GridSearchCV enabled us to greatly improve our mean cross validation r^2 score for ElasticNet() from 0.830 -> 0.866.

In [115]:
```
res = pd.pivot_table(pd.DataFrame(grid4.cv_results_), values='mean_test_score'
, index='param_elasticnet__alpha', columns= 'param_elasticnet__l1_ratio')

ax = sns.heatmap(res)
ax.set_title('Validation Score based on Alpha, l1 ratio parameters')
```

Out[115]:
```
Text(0.5,1,'Validation Score based on Alpha, l1 ratio parameters')
```



1.6: Visualize coefficients of resulting models

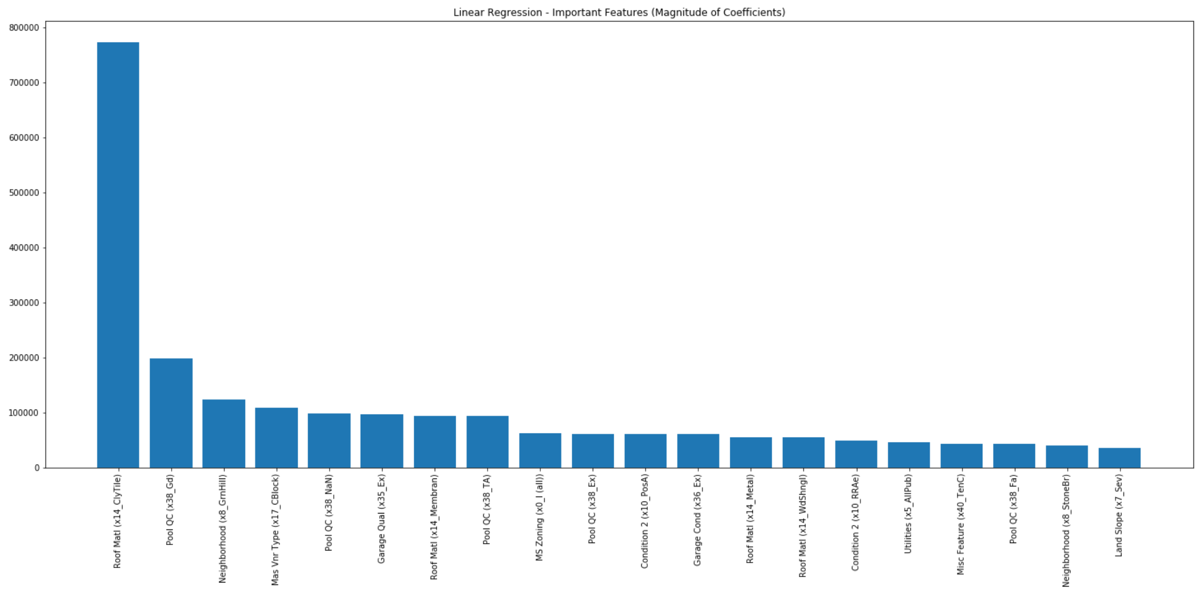
Linear Regression

In [154]:
```
from sklearn.preprocessing import OneHotEncoder

# Retrieve coef
model_opt = make_pipeline(preprocess, SimpleImputer(strategy = 'median'),
                          LinearRegression(fit_intercept = True, normalize = T
rue))
lr_coef = model_opt.fit(X_train, y_train).named_steps['linearregression'].coef
_

# Sort by highest magnitude coef
df_lr = pd.DataFrame(np.absolute(lr_coef)).sort_values(by = [0], ascending = F
alse).head(20)

#Rerun onehotencoder
h = OneHotEncoder(handle_unknown='ignore').fit(X_train.loc[:,cat_col])
h_df = pd.DataFrame(h.get_feature_names())
cont_col_df = pd.DataFrame(cont_col)

# Merge cont col names with one hot encoder col names
result = pd.concat([cont_col_df, h_df])
assert len(result)== len(lr_coef)
df_lr.index.name = 'index'
result.index = list(range(len(result)))
result.index.name = 'index'

# Merge
merged_lr = df_lr.merge(result, left_on='index', right_on = 'index', how = 'in
ner')
merged_lr['column_name'] = [cat_col[int(i.split('_')[0][1:])] for i in merged_
lr['0_y']]
merged_lr['column_name_join'] = merged_lr['column_name'] + ' (' + merged_lr['0
_y'] + ')'
merged_lr
```

Out[154]:

index	0_x	0_y	column_name	column_name_join
127	772439.692712	x14_ClyTile	Roof Matl	Roof Matl (x14_ClyTile)
283	198779.406420	x38_Gd	Pool QC	Pool QC (x38_Gd)
74	123662.760384	x8_GrnHill	Neighborhood	Neighborhood (x8_GrnHill)
167	108654.893632	x17_CBlock	Mas Vnr Type	Mas Vnr Type (x17_CBlock)
284	98365.697277	x38_NaN	Pool QC	Pool QC (x38_NaN)
266	96087.961476	x35_Ex	Garage Qual	Garage Qual (x35_Ex)
129	93537.824974	x14_Membran	Roof Matl	Roof Matl (x14_Membran)
285	93177.586825	x38_TA	Pool QC	Pool QC (x38_TA)
36	62822.150009	x0_I (all)	MS Zoning	MS Zoning (x0_I (all))
281	61361.020236	x38_Ex	Pool QC	Pool QC (x38_Ex)
103	61191.976411	x10_PosA	Condition 2	Condition 2 (x10_PosA)
272	60344.902659	x36_Ex	Garage Cond	Garage Cond (x36_Ex)
130	54962.709268	x14_Metal	Roof Matl	Roof Matl (x14_Metal)
133	54674.911413	x14_WdShngl	Roof Matl	Roof Matl (x14_WdShngl)
105	49240.760459	x10_RRAe	Condition 2	Condition 2 (x10_RRAe)
53	45696.047708	x5_AllPub	Utilities	Utilities (x5_AllPub)
295	43474.855959	x40_TenC	Misc Feature	Misc Feature (x40_TenC)
282	43474.855959	x38_Fa	Pool QC	Pool QC (x38_Fa)
88	40152.549868	x8_StoneBr	Neighborhood	Neighborhood (x8_StoneBr)
63	36118.262591	x7_Sev	Land Slope	Land Slope (x7_Sev)

In [192]:
```
fig = plt.figure(figsize=(25, 10))
ax = plt.gca()

plt.bar(x = merged_lr['column_name_join'], height = merged_lr['0_x'])
plt.title('Linear Regression - Important Features (Magnitude of Coefficients)'
)
plt.xticks(rotation=90)
plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

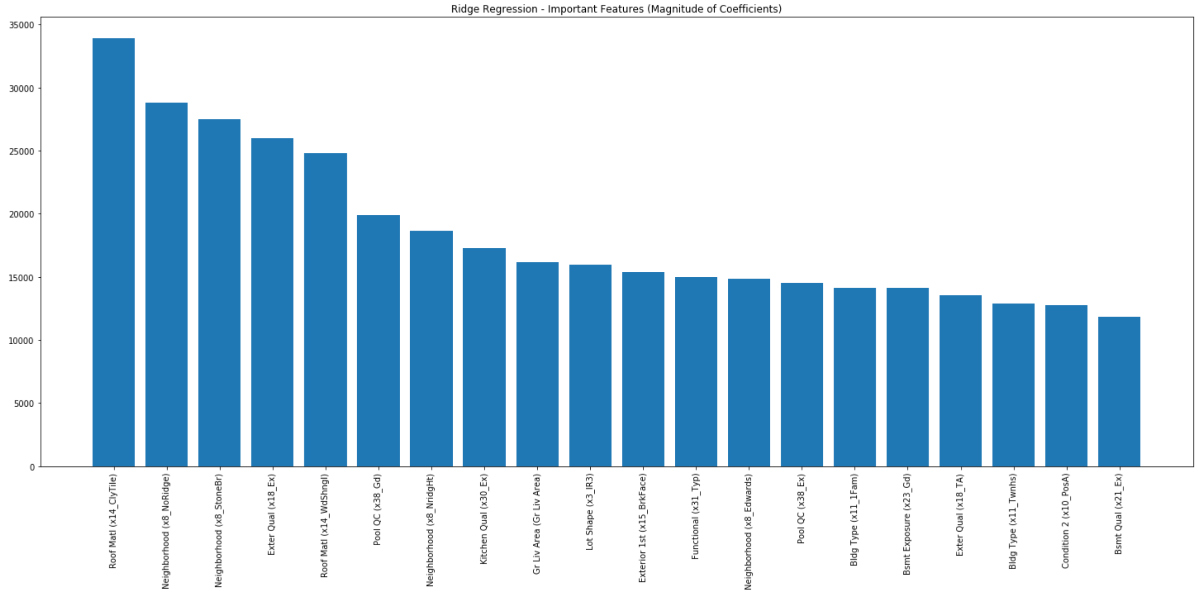
Ridge Regression

In [140]:
```
# Retrieve coef
model_opt2 = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), Rid
ge(alpha = 10.0))
rr_coef = model_opt2.fit(X_train, y_train).named_steps['ridge'].coef_

# Sort by highest magnitude coef
df_rr = pd.DataFrame(np.absolute(rr_coef)).sort_values(by = [0], ascending = F
alse).head(20)

#Rerun onehotencoder
h = OneHotEncoder(handle_unknown='ignore').fit(X_train.loc[:,cat_col])
h_df = pd.DataFrame(h.get_feature_names())
cont_col_df = pd.DataFrame(cont_col)

# Merge cont col names with one hot encoder col names
result = pd.concat([cont_col_df, h_df])
assert len(result)== len(rr_coef)
df_rr.index.name = 'index'
result.index = list(range(len(result)))
result.index.name = 'index'

# Merge
merged_rr = df_rr.merge(result, left_on='index', right_on = 'index', how = 'in
ner')

a = []

for i in merged_rr['0_y']:
    if i.startswith('x'):
        a.append(cat_col[int(i.split('_')[0][1:])])
    else:
        a.append(i)

merged_rr['column_name'] = a

merged_rr['column_name_join'] = merged_rr['column_name'] + ' (' + merged_rr['0
_y'] + ')'
merged_rr
```

Out[140]:

index	0_x	0_y	column_name	column_name_join
127	33909.457948	x14_ClyTile	Roof Matl	Roof Matl (x14_ClyTile)
81	28780.143188	x8_NoRidge	Neighborhood	Neighborhood (x8_NoRidge)
88	27508.533584	x8_StoneBr	Neighborhood	Neighborhood (x8_StoneBr)
171	25967.767335	x18_Ex	Exter Qual	Exter Qual (x18_Ex)
133	24790.064168	x14_WdShngl	Roof Matl	Roof Matl (x14_WdShngl)
283	19909.132216	x38_Gd	Pool QC	Pool QC (x38_Gd)
82	18663.908387	x8_NridgHt	Neighborhood	Neighborhood (x8_NridgHt)
236	17294.622017	x30_Ex	Kitchen Qual	Kitchen Qual (x30_Ex)
12	16194.201960	Gr Liv Area	Gr Liv Area	Gr Liv Area (Gr Liv Area)
47	15951.227073	x3_IR3	Lot Shape	Lot Shape (x3_IR3)
137	15358.290249	x15_BrkFace	Exterior 1st	Exterior 1st (x15_BrkFace)
248	14990.903837	x31_Typ	Functional	Functional (x31_Typ)
71	14824.165738	x8_Edwards	Neighborhood	Neighborhood (x8_Edwards)
281	14548.912042	x38_Ex	Pool QC	Pool QC (x38_Ex)
108	14155.003534	x11_1Fam	Bldg Type	Bldg Type (x11_1Fam)
199	14117.941987	x23_Gd	Bsmt Exposure	Bsmt Exposure (x23_Gd)
174	13574.534583	x18_TA	Exter Qual	Exter Qual (x18_TA)
111	12875.295590	x11_Twnhs	Bldg Type	Bldg Type (x11_Twnhs)
103	12791.677118	x10_PosA	Condition 2	Condition 2 (x10_PosA)
186	11851.079187	x21_Ex	Bsmt Qual	Bsmt Qual (x21_Ex)

In [190]:
```
fig = plt.figure(figsize=(25, 10))
ax = plt.gca()

plt.bar(x = merged_rr['column_name_join'], height = merged_rr['0_x'])
plt.title('Ridge Regression - Important Features (Magnitude of Coefficients)')
plt.xticks(rotation=90)
plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

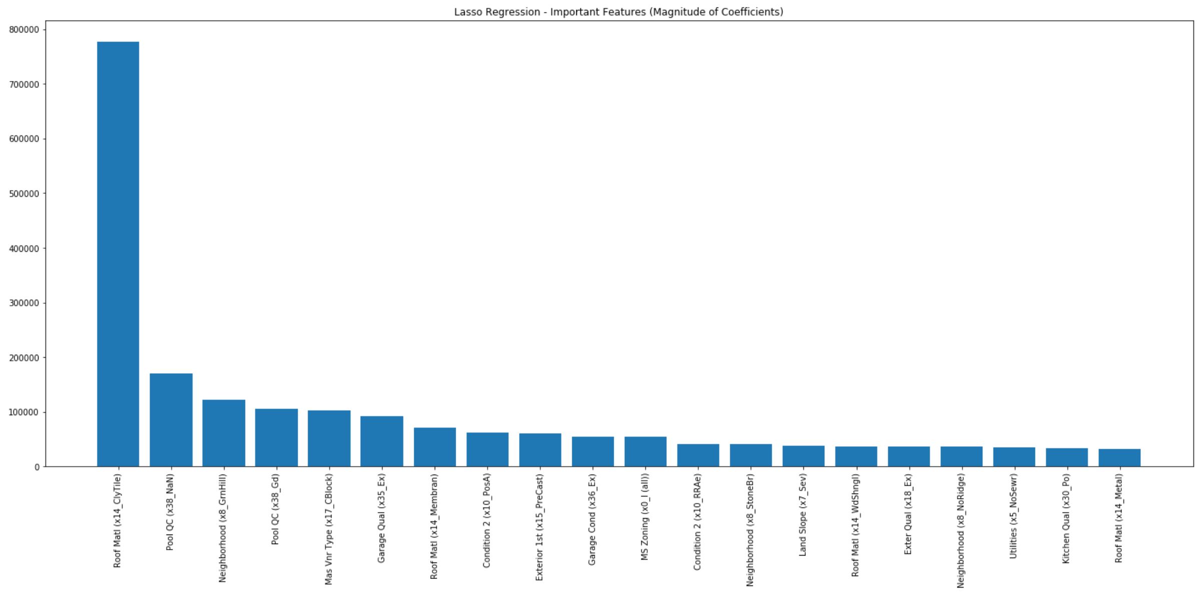
Lasso Regression

In [144]:
```
# Retrieve coef
model_opt3 = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), Las
so(alpha = 1.0))
lasso_coef = model_opt3.fit(X_train, y_train).named_steps['lasso'].coef_

# Sort by highest magnitude coef
df_lasso = pd.DataFrame(np.absolute(lasso_coef)).sort_values(by = [0], ascendi
ng = False).head(20)

#Rerun onehotencoder
h = OneHotEncoder(handle_unknown='ignore').fit(X_train.loc[:,cat_col])
h_df = pd.DataFrame(h.get_feature_names())
cont_col_df = pd.DataFrame(cont_col)

# Merge cont col names with one hot encoder col names
result = pd.concat([cont_col_df, h_df])
assert len(result)== len(lasso_coef)
df_lasso.index.name = 'index'
result.index = list(range(len(result)))
result.index.name = 'index'

# Merge
merged_lasso = df_lasso.merge(result, left_on='index', right_on = 'index', how
= 'inner')
merged_lasso['column_name'] = [cat_col[int(i.split('_')[0][1:])] for i in merg
ed_lasso['0_y']]
merged_lasso['column_name_join'] = merged_lasso['column_name'] + ' (' + merged
_lasso['0_y'] + ')'
merged_lasso
```

Out[144]:

	0_x	0_y	column_name	column_name_join
index				
127	777699.031832	x14_ClyTile	Roof Matl	Roof Matl (x14_ClyTile)
284	170939.853528	x38_NaN	Pool QC	Pool QC (x38_NaN)
74	122086.581171	x8_GrnHill	Neighborhood	Neighborhood (x8_GrnHill)
283	105768.412763	x38_Gd	Pool QC	Pool QC (x38_Gd)
167	103268.179307	x17_CBlock	Mas Vnr Type	Mas Vnr Type (x17_CBlock)
266	91495.135872	x35_Ex	Garage Qual	Garage Qual (x35_Ex)
129	70739.694262	x14_Membran	Roof Matl	Roof Matl (x14_Membran)
103	62779.739448	x10_PosA	Condition 2	Condition 2 (x10_PosA)
144	59977.707018	x15_PreCast	Exterior 1st	Exterior 1st (x15_PreCast)
272	54444.235003	x36_Ex	Garage Cond	Garage Cond (x36_Ex)
36	53993.581245	x0_I (all)	MS Zoning	MS Zoning (x0_I (all))
105	40977.214013	x10_RRAe	Condition 2	Condition 2 (x10_RRAe)
88	40518.432799	x8_StoneBr	Neighborhood	Neighborhood (x8_StoneBr)
63	37587.047166	x7_Sev	Land Slope	Land Slope (x7_Sev)
133	37016.267788	x14_WdShngl	Roof Matl	Roof Matl (x14_WdShngl)
171	36394.033097	x18_Ex	Exter Qual	Exter Qual (x18_Ex)
81	36015.872959	x8_NoRidge	Neighborhood	Neighborhood (x8_NoRidge)
55	35567.613543	x5_NoSewr	Utilities	Utilities (x5_NoSewr)
239	32917.769627	x30_Po	Kitchen Qual	Kitchen Qual (x30_Po)
130	31438.668152	x14_Metal	Roof Matl	Roof Matl (x14_Metal)

In [189]:
```
fig = plt.figure(figsize=(25, 10))
ax = plt.gca()

plt.bar(x = merged_lasso['column_name_join'], height = merged_lasso['0_x'])
plt.title('Lasso Regression - Important Features (Magnitude of Coefficients)')
plt.xticks(rotation=90)
plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

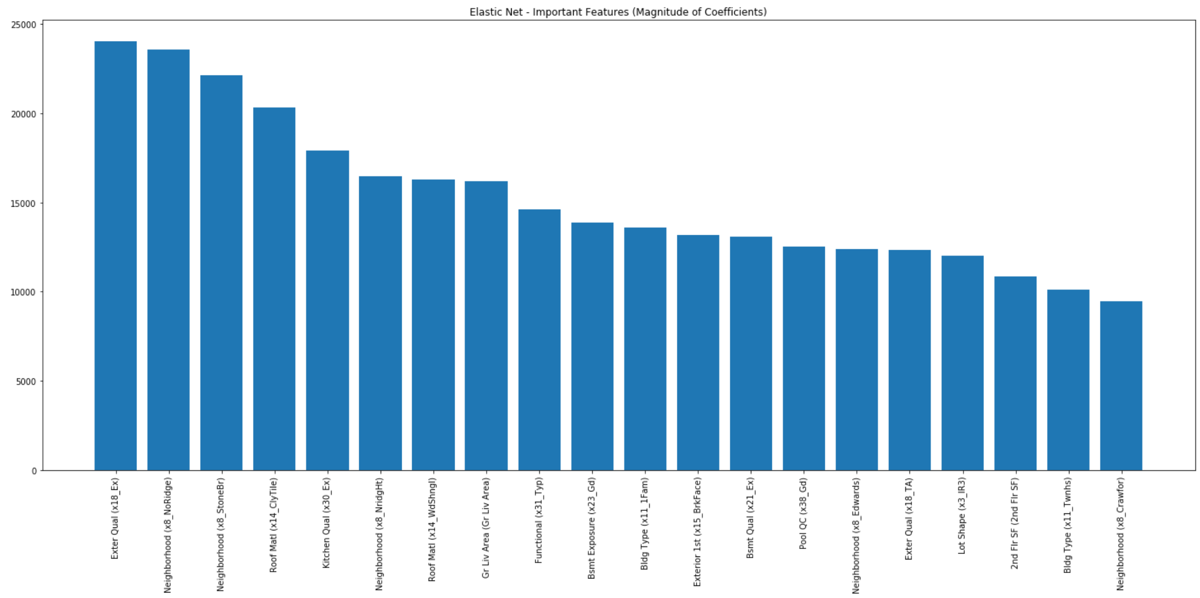
Elastic Net

In [148]:
```
# Retrieve coef
model_opt4 = make_pipeline(preprocess, SimpleImputer(strategy = 'median'),
                           ElasticNet(alpha= 0.01, l1_ratio = 0.1))
en_coef = model_opt4.fit(X_train, y_train).named_steps['elasticnet'].coef_

# Sort by highest magnitude coef
df_en = pd.DataFrame(np.absolute(en_coef)).sort_values(by = [0], ascending = F
alse).head(20)

#Rerun onehotencoder
h = OneHotEncoder(handle_unknown='ignore').fit(X_train.loc[:,cat_col])
h_df = pd.DataFrame(h.get_feature_names())
cont_col_df = pd.DataFrame(cont_col)

# Merge cont col names with one hot encoder col names
result = pd.concat([cont_col_df, h_df])
assert len(result)== len(en_coef)
df_en.index.name = 'index'
result.index = list(range(len(result)))
result.index.name = 'index'

# Merge
merged_en = df_en.merge(result, left_on='index', right_on = 'index', how = 'in
ner')
a = []

for i in merged_en['0_y']:
    if i.startswith('x'):
        a.append(cat_col[int(i.split('_')[0][1:])])
    else:
        a.append(i)

merged_en['column_name'] = a
merged_en['column_name_join'] = merged_en['column_name'] + ' (' + merged_en['0
_y'] + ')'
merged_en
```

Out[148]:

index	0_x	0_y	column_name	column_name_join
171	24023.591226	x18_Ex	Exter Qual	Exter Qual (x18_Ex)
81	23575.213457	x8_NoRidge	Neighborhood	Neighborhood (x8_NoRidge)
88	22139.455827	x8_StoneBr	Neighborhood	Neighborhood (x8_StoneBr)
127	20299.424367	x14_ClyTile	Roof Matl	Roof Matl (x14_ClyTile)
236	17894.557191	x30_Ex	Kitchen Qual	Kitchen Qual (x30_Ex)
82	16453.641899	x8_NridgHt	Neighborhood	Neighborhood (x8_NridgHt)
133	16287.141161	x14_WdShngl	Roof Matl	Roof Matl (x14_WdShngl)
12	16192.115042	Gr Liv Area	Gr Liv Area	Gr Liv Area (Gr Liv Area)
248	14589.437814	x31_Typ	Functional	Functional (x31_Typ)
199	13884.565293	x23_Gd	Bsmt Exposure	Bsmt Exposure (x23_Gd)
108	13576.443242	x11_1Fam	Bldg Type	Bldg Type (x11_1Fam)
137	13170.289281	x15_BrkFace	Exterior 1st	Exterior 1st (x15_BrkFace)
186	13064.397584	x21_Ex	Bsmt Qual	Bsmt Qual (x21_Ex)
283	12515.274592	x38_Gd	Pool QC	Pool QC (x38_Gd)
71	12386.935222	x8_Edwards	Neighborhood	Neighborhood (x8_Edwards)
174	12326.934205	x18_TA	Exter Qual	Exter Qual (x18_TA)
47	12008.275669	x3_IR3	Lot Shape	Lot Shape (x3_IR3)
10	10853.553831	2nd Flr SF	2nd Flr SF	2nd Flr SF (2nd Flr SF)
111	10098.169507	x11_Twnhs	Bldg Type	Bldg Type (x11_Twnhs)
70	9478.653212	x8_Crawfor	Neighborhood	Neighborhood (x8_Crawfor)

In [191]:
```
fig = plt.figure(figsize=(25, 10))
ax = plt.gca()

plt.bar(x = merged_en['column_name_join'], height = merged_en['0_x'])
plt.title('Elastic Net - Important Features (Magnitude of Coefficients)')
plt.xticks(rotation=90)
plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

1.6 Insights:

Originally, we found through inital exploratory data visualization that the top three categorical variables would be 'Neighborhood', 'Bsmt Qual', and 'External Qual'. After going through the modeling process, our four different models have revealed other variables that may also play an significant role in predicting 'SalePrice'. The majority of these important variables from the model results seem to be categorical variables. Examining all four models, we see significant overlap in the results across the models. For instance, 'Exter Qual', 'Neighborhood', and 'Roof Matl', 'Pool QC' were within the top 10-20 variables for all of the models. What was especially interesting was seeing just how much emphasis and weight each model placed on these variables and others based on the optimal parameters we discovered using GridSearch. For instance, while most of the models had pretty large (in a decreasing manner) coefficients assigned to the varialbes, Linear Regression and Lasso Regression both had designated 'Roof Matl' as being the most influential, while the other variables in the chart had much smaller magnitude of coefficients. If you take a look at the Lasso or Linear Regression variable visualization, there is a huge drop from the 'Roof Matl' coefficient compared with other variables in the top 10-20. Given the model results, the results are close to what we saw when we plotted the Top 3 categorical variables in our initial exploratory data visualization.

Homework 2 - Task 2

Michelle Chen (mc4571)

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [41]:
```
tel = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
tel = tel.drop(columns = ['customerID'])
y = tel['Churn']
tel = tel.iloc[:,:-1]
tel.head()
```

Out[41]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServi
0	Female	0	Yes	No	1	No	No phone service	D
1	Male	0	No	No	34	Yes	No	D
2	Male	0	No	No	2	Yes	No	D
3	Male	0	No	No	45	No	No phone service	D
4	Female	0	No	No	2	Yes	No	Fiber op

In [42]:
```
tel.shape
```

Out[42]: (7043, 19)

2.1: Visualize univariate distribution

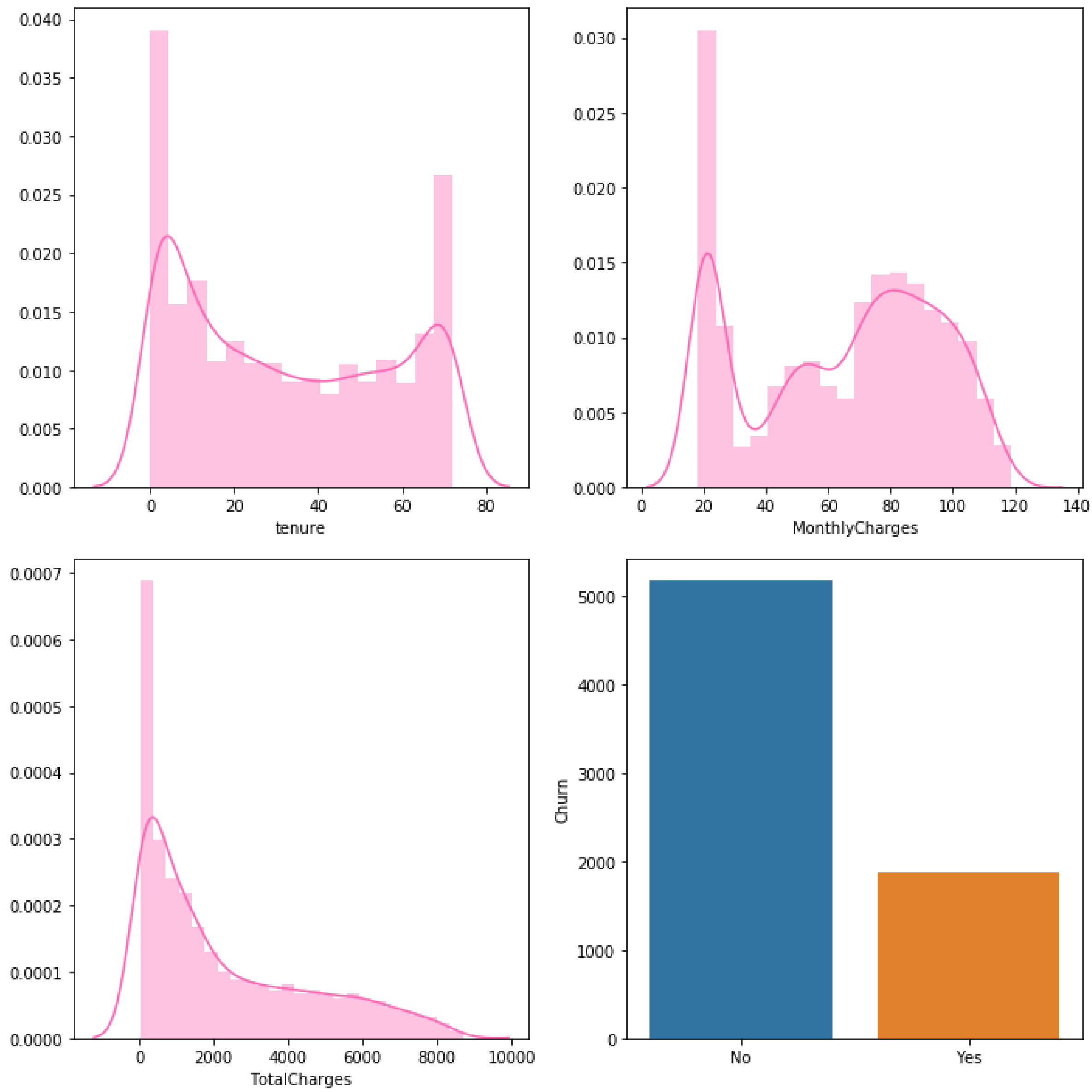
In [43]:
```
tel['SeniorCitizen']= tel['SeniorCitizen'].astype('object')
tel['TotalCharges'] = pd.to_numeric(tel['TotalCharges'], errors='coerce')

categorical = tel.dtypes == object
cat_col = list(categorical[categorical == True].index)

cont = tel.dtypes != object
cont_col = list(cont[cont == True].index)

cat_col
```

Out[43]:
```
['gender',
 'SeniorCitizen',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod']
```

In [44]:
```
cont_col
```

Out[44]:
```
['tenure', 'MonthlyCharges', 'TotalCharges']
```

In [46]:
```
fig, ax = plt.subplots(2,2,figsize=(10, 10))

k = 0

for i in range(0,2):
    for j in range(0,2):
        if k < 3:
            sns.distplot(tel[cont_col[k]][~np.isnan(tel[cont_col[k]])], color
= 'hotpink', ax = ax[i,j])
            k+=1
        else:
            sns.barplot(x = y.value_counts().index, y = y.value_counts(), ax =
ax[i,j])



plt.tight_layout()
```



2.2: Split data and build pipeline

In [57]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tel, y, random_state = 0,s
tratify = y)
```

In [59]:
```
X_train.head()
print(X_train.shape, X_test.shape)
```

```
(5282, 19) (1761, 19)
```

Logistic Regression Pipeline

In [71]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.impute import SimpleImputer
```

In [72]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
```

In [73]:
```
# Nonstandardized
preprocess_nonscale = make_column_transformer((SimpleImputer(strategy = 'media
n'), cont_col,),
                                              (OneHotEncoder(handle_unknown='i
gnore'), cat_col))
model = make_pipeline(preprocess_nonscale, LogisticRegression())
predicted_nonscale_logr = cross_val_score(model, X_train, y_train, cv=5, scori
ng = 'accuracy')
print('No Standard Scaler():', np.mean(predicted_nonscale_logr))

# Standardized
preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'),LogisticR
egression())
predicted_logr = cross_val_score(model, X_train, y_train, cv=5, scoring = 'acc
uracy')
print('Standard Scaler():', np.mean(predicted_logr))
```

```
No Standard Scaler(): 0.8085960121556148
Standard Scaler(): 0.8087859436369371
```

How does scaling influence results (Logr)?

There is a very slight boost in our mean cross validated accuracy measure, however in general scaling does not seem to influence the results very much. This may change if we increase our number of cross

Linear Support Vector Machine

In [74]:
```
from sklearn.svm import LinearSVC
```

In [75]:
```
# Nonstandardized
preprocess_nonscale = make_column_transformer((SimpleImputer(strategy = 'media
n'), cont_col,),
                                              (OneHotEncoder(handle_unknown='i
gnore'), cat_col))
model = make_pipeline(preprocess_nonscale, LinearSVC())
predicted_nonscale_lsvc = cross_val_score(model, X_train, y_train, cv=5, scori
ng = 'accuracy')
print('No Standard Scaler():', np.mean(predicted_nonscale_lsvc))


# Standardized
preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), LinearSV
C())
predicted_lsvc = cross_val_score(model, X_train, y_train, cv=5, scoring = 'acc
uracy')
print('StandardScaler():', np.mean(predicted_lsvc))
```

```
No Standard Scaler(): 0.692512220119836
StandardScaler(): 0.8048101043548064
```

How does scaling influence results (Lin SVM)?

Scaling actually improves the classification accuracy dramatically.

Nearest Centroid

In [76]:
```
from sklearn.neighbors.nearest_centroid import NearestCentroid
from sklearn.model_selection import cross_val_score
```

In [77]:
```
# Nonstandardized

preprocess_nonscale = make_column_transformer((SimpleImputer(strategy = 'media
n'), cont_col),
                                              (OneHotEncoder(handle_unknown='i
gnore'), cat_col))
model = make_pipeline(preprocess_nonscale, NearestCentroid())
predicted_nonscale_nc = cross_val_score(model, X_train, y_train, cv=5, scoring
= 'accuracy')
print('No Standard Scaler():', np.mean(predicted_nonscale_nc))

# Standardized

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), NearestC
entroid())
predicted_nc = cross_val_score(model, X_train, y_train, cv=5, scoring = 'accur
acy')
print('StandardScaler():', np.mean(predicted_nc))
```

```
No Standard Scaler(): 0.5153346377684127
StandardScaler(): 0.735700847166079
```

How does scaling influence results (Logr)?

Scaling improves our accuracy drastically.

2.3: Tune parameters using GridSearchCV

In [78]:
```
from sklearn.model_selection import GridSearchCV
```

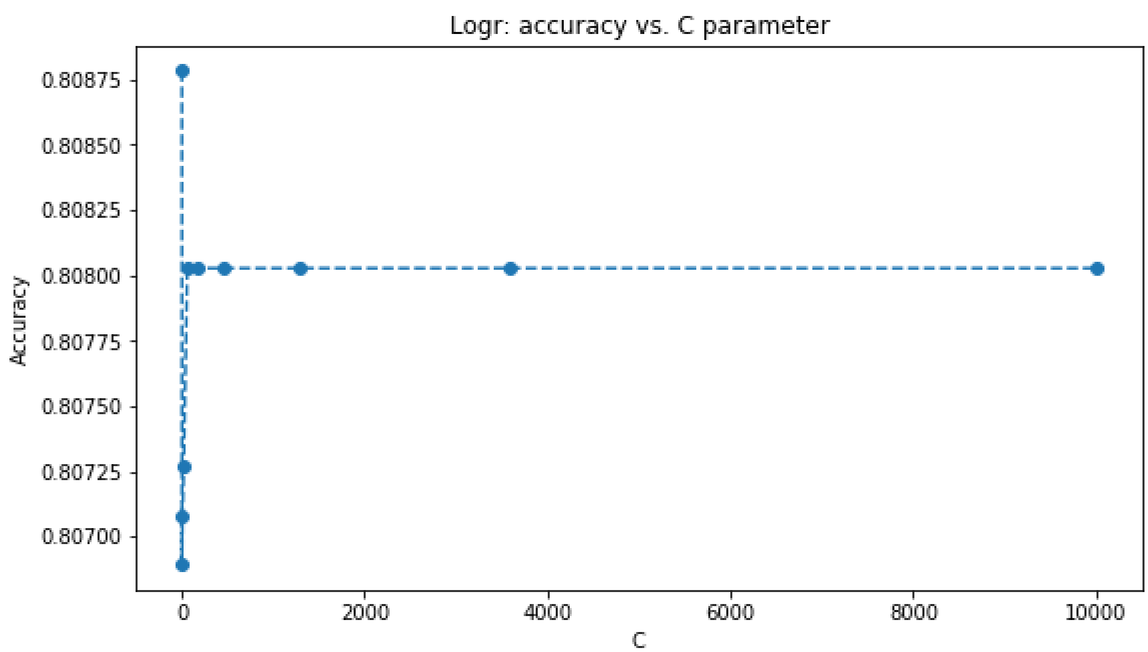
LR

In [79]:
```
param_grid_logr = {'logisticregression__C': np.logspace(0, 4, 10)}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), Logistic
Regression())

grid_logr = GridSearchCV(model, param_grid=param_grid_logr, cv=5, scoring = 'a
ccuracy')

grid_logr.fit(X_train, y_train)

print("Logr - best mean cross-validation score: {:.3f}".format(grid_logr.best_
score_))
print("Logr - best parameters: {}".format(grid_logr.best_params_))
```

```
Logr - best mean cross-validation score: 0.809
Logr - best parameters: {'logisticregression__C': 1.0}
```

In [80]:
```
# visualize performance as a function of parameters

x_score = pd.DataFrame(grid_logr.cv_results_['mean_test_score'])
parameters = pd.DataFrame(param_grid_logr['logisticregression__C'])

fig = plt.figure(figsize=(9, 5))
ax = plt.gca()

ax.plot(parameters, x_score, marker='o', linestyle='dashed')
ax.set_title('Logr: accuracy vs. C parameter')
ax.set_xlabel('C')
ax.set_ylabel('Accuracy')
```

Out[80]:
```
Text(0,0.5,'Accuracy')
```



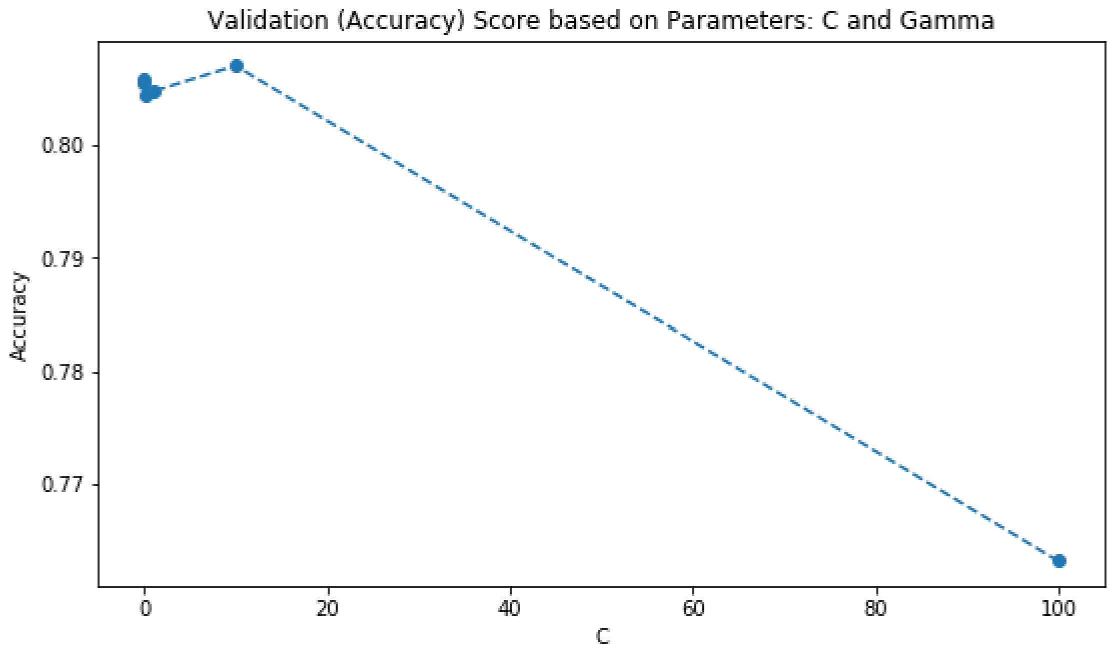
LSVM

In [81]:
```
param_grid_linSVC = {'linearsvc__C': np.logspace(-3, 2, 6)}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), LinearSV
C())
sorted(model.get_params().keys())
grid_linSVC = GridSearchCV(model, param_grid=param_grid_linSVC,
                    cv=5, scoring = 'accuracy')

grid_linSVC.fit(X_train, y_train)

print("Linear SVC - best mean cross-validation score: {:.3f}".format(grid_linS
VC.best_score_))
print("Linear SVC - best parameters: {}".format(grid_linSVC.best_params_))
```

```
Linear SVC - best mean cross-validation score: 0.807
Linear SVC - best parameters: {'linearsvc__C': 10.0}
```

In [82]:
```
x_score = pd.DataFrame(grid_linSVC.cv_results_['mean_test_score'])
parameters = pd.DataFrame(param_grid_linSVC['linearsvc__C'])

fig = plt.figure(figsize=(9, 5))
ax = plt.gca()

ax.plot(parameters, x_score, marker='o', linestyle='dashed')
ax.set_xlabel('C')
ax.set_ylabel('Accuracy')
ax.set_title('Validation (Accuracy) Score based on Parameters: C and Gamma')
```

Out[82]:
```
Text(0.5,1,'Validation (Accuracy) Score based on Parameters: C and Gamma')
```



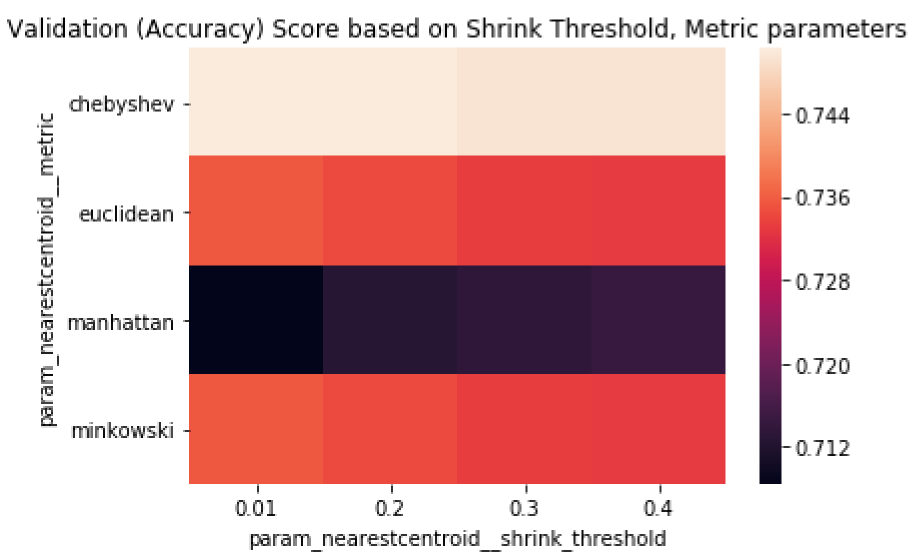
Centroid

In [83]:
```
param_grid_centroid = {'nearestcentroid__metric': ('euclidean', 'manhattan',
'chebyshev', 'minkowski'),
                       'nearestcentroid__shrink_threshold': (None, 0.01, 0.2,
0.3, 0.4)}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), NearestC
entroid())
#sorted(model.get_params().keys())
grid_nc = GridSearchCV(model, param_grid=param_grid_centroid,
                    cv=5, scoring = 'accuracy')

grid_nc.fit(X_train, y_train)

print("Nearest Centroid - best mean cross-validation score: {:.3f}".format(gri
d_nc.best_score_))
print("Nearest Centroid - best parameters: {}".format(grid_nc.best_params_))
```

```
Nearest Centroid - best mean cross-validation score: 0.750
Nearest Centroid - best parameters: {'nearestcentroid__metric': 'chebyshev',
'nearestcentroid__shrink_threshold': None}
```

In [84]:
```
res = pd.pivot_table(pd.DataFrame(grid_nc.cv_results_), values='mean_test_scor
e', index='param_nearestcentroid__metric', columns= 'param_nearestcentroid__sh
rink_threshold')

ax = sns.heatmap(res)
ax.set_title('Validation (Accuracy) Score based on Shrink Threshold, Metric pa
rameters')
```

Out[84]:
```
Text(0.5,1,'Validation (Accuracy) Score based on Shrink Threshold, Metric par
ameters')
```



2.4: Change from stratified k-fold to kfold with shuffling

In [86]:
```
from sklearn.model_selection import KFold
```

In [126]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tel, y, random_state = 0)
```

Logistic Regression

In [127]:
```
param_grid_logr = {'logisticregression__C': np.logspace(0, 4, 10)}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), Logistic
Regression())

grid_logr = GridSearchCV(model, param_grid=param_grid_logr, cv = KFold(shuffle
=True), scoring = 'accuracy')

grid_logr.fit(X_train, y_train)

print("Logr - best mean cross-validation score: {:.3f}".format(grid_logr.best_
score_))
print("Logr - best parameters: {}".format(grid_logr.best_params_))
```

```
Logr - best mean cross-validation score: 0.805
Logr - best parameters: {'logisticregression__C': 1.0}
```

In [128]:
```
grid_logr = GridSearchCV(model, param_grid=param_grid_logr, cv = KFold(shuffle
=True, random_state = 0), scoring = 'accuracy')

grid_logr.fit(X_train, y_train)

print("Logr - best mean cross-validation score: {:.3f}".format(grid_logr.best_
score_))
print("Logr - best parameters: {}".format(grid_logr.best_params_))
```

```
Logr - best mean cross-validation score: 0.807
Logr - best parameters: {'logisticregression__C': 59.94842503189409}
```

In [129]:
```
grid_logr = GridSearchCV(model, param_grid=param_grid_logr, cv = KFold(shuffle
=True, random_state = 0, n_splits = 2), scoring = 'accuracy')

grid_logr.fit(X_train, y_train)

print("Logr - best mean cross-validation score: {:.3f}".format(grid_logr.best_
score_))
print("Logr - best parameters: {}".format(grid_logr.best_params_))
```

```
Logr - best mean cross-validation score: 0.806
Logr - best parameters: {'logisticregression__C': 21.544346900318832}
```

In [130]:
```
grid_logr = GridSearchCV(model, param_grid=param_grid_logr, cv = KFold(shuffle
=True, random_state = 0, n_splits = 3), scoring = 'accuracy')

grid_logr.fit(X_train, y_train)

print("Logr - best mean cross-validation score: {:.3f}".format(grid_logr.best_
score_))
print("Logr - best parameters: {}".format(grid_logr.best_params_))
```

```
Logr - best mean cross-validation score: 0.807
Logr - best parameters: {'logisticregression__C': 59.94842503189409}
```

Insights: KFold did not help our accuracy considerably, in fact our mean cross validation accuracy score decreased for every one of the above iterations when we set a random state, increased n_splits, and set shuffle = True. The C parameter, however, did change when we set a random state and designated a n_splits values, in addition to having set shuffle = True.

Linear SVM

In [131]:
```
param_grid_linSVC = {'linearsvc__C': np.logspace(-3, 2, 6)}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), LinearSV
C())

grid_linSVC = GridSearchCV(model, param_grid=param_grid_linSVC,
                    cv= KFold(shuffle = True), scoring = 'accuracy')

grid_linSVC.fit(X_train, y_train)

print("Linear SVC - best mean cross-validation score: {:.3f}".format(grid_linS
VC.best_score_))
print("Linear SVC - best parameters: {}".format(grid_linSVC.best_params_))
```

```
Linear SVC - best mean cross-validation score: 0.806
Linear SVC - best parameters: {'linearsvc__C': 0.001}
```

In [132]:
```
grid_linSVC = GridSearchCV(model, param_grid=param_grid_linSVC,
                    cv= KFold(shuffle = True, random_state = 0), scoring = 'ac
curacy')

grid_linSVC.fit(X_train, y_train)

print("Linear SVC - best mean cross-validation score: {:.3f}".format(grid_linS
VC.best_score_))
print("Linear SVC - best parameters: {}".format(grid_linSVC.best_params_))
```

```
Linear SVC - best mean cross-validation score: 0.807
Linear SVC - best parameters: {'linearsvc__C': 10.0}
```

In [134]:
```
grid_linSVC = GridSearchCV(model, param_grid=param_grid_linSVC,
                    cv= KFold(shuffle = True, random_state = 0, n_splits = 3),
scoring = 'accuracy')

grid_linSVC.fit(X_train, y_train)

print("Linear SVC - best mean cross-validation score: {:.3f}".format(grid_linS
VC.best_score_))
print("Linear SVC - best parameters: {}".format(grid_linSVC.best_params_))
```

```
Linear SVC - best mean cross-validation score: 0.803
Linear SVC - best parameters: {'linearsvc__C': 0.001}
```

Insights: When using Kfold instead of Stratified Kfold, the parameters did change when we set designated a n_splits values and when we set shuffle = True. We got an equivalent parameter to the 2.3 Linear SVC best parameter when we set a random_state = 0 and shuffle = True. I presume that depending on the random_state we choose, we are also likely to get other optimal C parameters.

Centroid

In [135]:
```
param_grid_centroid = {'nearestcentroid__metric': ('euclidean', 'manhattan',
'chebyshev', 'minkowski'),
                       'nearestcentroid__shrink_threshold': (None, 0.01, 0.2,
0.3, 0.4)}

preprocess = make_column_transformer(
    (StandardScaler(), cont_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
model = make_pipeline(preprocess, SimpleImputer(strategy = 'median'), NearestC
entroid())

grid_nc = GridSearchCV(model, param_grid=param_grid_centroid,
                    cv= KFold(shuffle = True), scoring = 'accuracy')

grid_nc.fit(X_train, y_train)

print("Nearest Centroid - best mean cross-validation score: {:.3f}".format(gri
d_nc.best_score_))
print("Nearest Centroid - best parameters: {}".format(grid_nc.best_params_))
```

```
Nearest Centroid - best mean cross-validation score: 0.755
Nearest Centroid - best parameters: {'nearestcentroid__metric': 'chebyshev',
'nearestcentroid__shrink_threshold': None}
```

In [136]:
```
grid_nc = GridSearchCV(model, param_grid=param_grid_centroid,
                    cv= KFold(shuffle = True, random_state = 0), scoring = 'ac
curacy')

grid_nc.fit(X_train, y_train)

print("Nearest Centroid - best mean cross-validation score: {:.3f}".format(gri
d_nc.best_score_))
print("Nearest Centroid - best parameters: {}".format(grid_nc.best_params_))
```

```
Nearest Centroid - best mean cross-validation score: 0.754
Nearest Centroid - best parameters: {'nearestcentroid__metric': 'chebyshev',
'nearestcentroid__shrink_threshold': None}
```

In [137]:
```
grid_nc = GridSearchCV(model, param_grid=param_grid_centroid,
                    cv= KFold(shuffle = True, random_state = 0, n_splits = 2),
scoring = 'accuracy')

grid_nc.fit(X_train, y_train)

print("Nearest Centroid - best mean cross-validation score: {:.3f}".format(gri
d_nc.best_score_))
print("Nearest Centroid - best parameters: {}".format(grid_nc.best_params_))
```

```
Nearest Centroid - best mean cross-validation score: 0.752
Nearest Centroid - best parameters: {'nearestcentroid__metric': 'chebyshev',
'nearestcentroid__shrink_threshold': None}
```

In [138]:
```
grid_nc = GridSearchCV(model, param_grid=param_grid_centroid,
                    cv = KFold(shuffle = True, random_state = 0, n_splits = 3
), scoring = 'accuracy')

grid_nc.fit(X_train, y_train)

print("Nearest Centroid - best mean cross-validation score: {:.3f}".format(gri
d_nc.best_score_))
print("Nearest Centroid - best parameters: {}".format(grid_nc.best_params_))
```

```
Nearest Centroid - best mean cross-validation score: 0.754
Nearest Centroid - best parameters: {'nearestcentroid__metric': 'chebyshev',
'nearestcentroid__shrink_threshold': None}
```

Insights: When using Kfold instead of Stratified Kfold, the parameters did not change even when we set designated a n_splits values, set shuffle = True, and set a random_state.

2.5: Visualize coefficients for LR and LSVM using hyperparameters that performed well in grid search

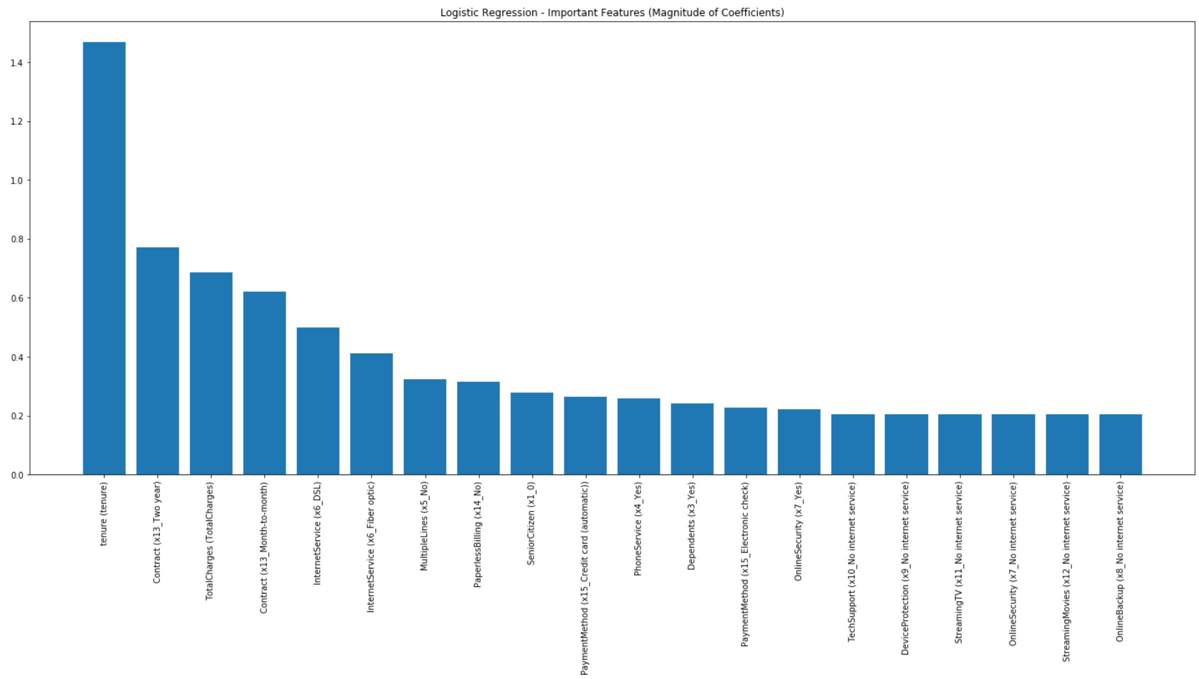
Logistic Regression

In [156]:
```
from sklearn.preprocessing import OneHotEncoder

# Retrieve coef
model_opt_logr = make_pipeline(preprocess, SimpleImputer(strategy = 'median'),
LogisticRegression(C = 59.94842503189409))
logr_coef = model_opt_logr.fit(X_train, y_train).named_steps['logisticregressi
on'].coef_[0]

# Sort by highest magnitude coef
df_logr = pd.DataFrame(np.absolute(logr_coef)).sort_values(by = [0], ascending
= False).head(20)

# Rerun onehotencoder
h = OneHotEncoder(handle_unknown='ignore').fit(X_train.loc[:,cat_col])
h_df = pd.DataFrame(h.get_feature_names())
cont_col_df = pd.DataFrame(cont_col)

# Merge cont col names with one hot encoder col names
result = pd.concat([cont_col_df, h_df])
assert len(result)== len(logr_coef)
df_logr.index.name = 'index'
result.index = list(range(len(result)))
result.index.name = 'index'

# Merge
merged_logr = df_logr.merge(result, left_on='index', right_on = 'index', how =
'inner')

a = []

for i in merged_logr['0_y']:
    if i.startswith('x'):
        a.append(cat_col[int(i.split('_')[0][1:])])
    else:
        a.append(i)

merged_logr['column_name'] = a
merged_logr['column_name_join'] = merged_logr['column_name'] + ' (' + merged_l
ogr['0_y'] + ')'
merged_logr
```

Out[156]:

	0_x	0_y	column_name	column_name_join
index				
0	1.466901	tenure	tenure	tenure (tenure)
39	0.772585	x13_Two year	Contract	Contract (x13_Two year)
2	0.687506	TotalCharges	TotalCharges	TotalCharges (TotalCharges)
37	0.622056	x13_Month-to-month	Contract	Contract (x13_Month-to-month)
16	0.499000	x6_DSL	InternetService	InternetService (x6_DSL)
17	0.411906	x6_Fiber optic	InternetService	InternetService (x6_Fiber optic)
13	0.322703	x5_No	MultipleLines	MultipleLines (x5_No)
40	0.316305	x14_No	PaperlessBilling	PaperlessBilling (x14_No)
5	0.278740	x1_0	SeniorCitizen	SeniorCitizen (x1_0)
43	0.264619	x15_Credit card (automatic)	PaymentMethod	PaymentMethod (x15_Credit card (automatic))
12	0.259730	x4_Yes	PhoneService	PhoneService (x4_Yes)
10	0.241958	x3_Yes	Dependents	Dependents (x3_Yes)
44	0.227614	x15_Electronic check	PaymentMethod	PaymentMethod (x15_Electronic check)
21	0.220374	x7_Yes	OnlineSecurity	OnlineSecurity (x7_Yes)
29	0.205200	x10_No internet service	TechSupport	TechSupport (x10_No internet service)
26	0.205200	x9_No internet service	DeviceProtection	DeviceProtection (x9_No internet service)
32	0.205200	x11_No internet service	StreamingTV	StreamingTV (x11_No internet service)
20	0.205200	x7_No internet service	OnlineSecurity	OnlineSecurity (x7_No internet service)
35	0.205200	x12_No internet service	StreamingMovies	StreamingMovies (x12_No internet service)
23	0.205200	x8_No internet service	OnlineBackup	OnlineBackup (x8_No internet service)

In [165]:
```
fig = plt.figure(figsize=(25, 10))
ax = plt.gca()

plt.bar(x = merged_logr['column_name_join'], height = merged_logr['0_x'])
plt.title('Logistic Regression - Important Features (Magnitude of Coefficient
s)')
plt.xticks(rotation=90)
plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

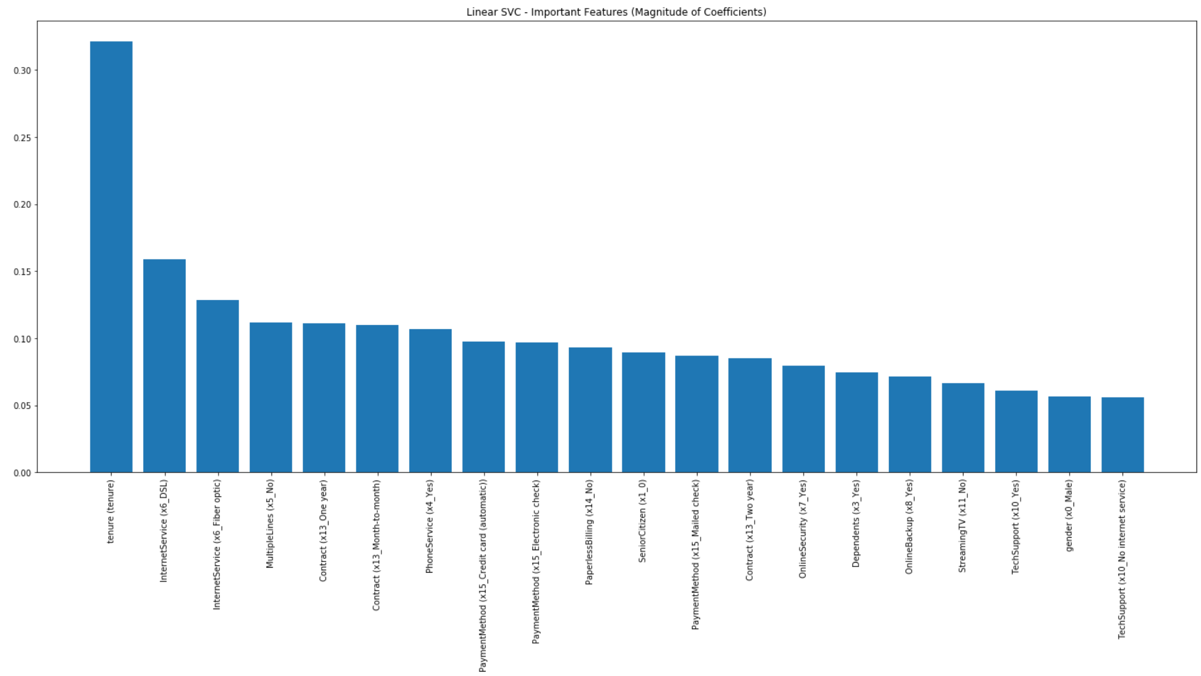
LinearSVC

In [161]:
```
# Retrieve coef
model_opt_linSVC = make_pipeline(preprocess, SimpleImputer(strategy = 'median'
), LinearSVC(C = 10))
svc_coef = model_opt_linSVC.fit(X_train, y_train).named_steps['linearsvc'].coe
f_[0]

# Sort by highest magnitude coef
df_svc = pd.DataFrame(np.absolute(svc_coef)).sort_values(by = [0], ascending =
False).head(20)

#Rerun onehotencoder
h = OneHotEncoder(handle_unknown='ignore').fit(X_train.loc[:,cat_col])
h_df = pd.DataFrame(h.get_feature_names())
cont_col_df = pd.DataFrame(cont_col)

# Merge cont col names with one hot encoder col names
result = pd.concat([cont_col_df, h_df])
assert len(result)== len(svc_coef)
df_svc.index.name = 'index'
result.index = list(range(len(result)))
result.index.name = 'index'

# Merge
merged_svc = df_svc.merge(result, left_on='index', right_on = 'index', how =
'inner')

a = []

for i in merged_svc['0_y']:
    if i.startswith('x'):
        a.append(cat_col[int(i.split('_')[0][1:])])
    else:
        a.append(i)

merged_svc['column_name'] = a
merged_svc['column_name_join'] = merged_svc['column_name'] + ' (' + merged_svc
['0_y'] + ')'
merged_svc
```

Out[161]:

index	0_x	0_y	column_name	column_name_join
0	0.321043	tenure	tenure	tenure (tenure)
16	0.159060	x6_DSL	InternetService	InternetService (x6_DSL)
17	0.128473	x6_Fiber optic	InternetService	InternetService (x6_Fiber optic)
13	0.111792	x5_No	MultipleLines	MultipleLines (x5_No)
38	0.111266	x13_One year	Contract	Contract (x13_One year)
37	0.110004	x13_Month-to-month	Contract	Contract (x13_Month-to-month)
12	0.106937	x4_Yes	PhoneService	PhoneService (x4_Yes)
43	0.097571	x15_Credit card (automatic)	PaymentMethod	PaymentMethod (x15_Credit card (automatic))
44	0.097007	x15_Electronic check	PaymentMethod	PaymentMethod (x15_Electronic check)
40	0.092989	x14_No	PaperlessBilling	PaperlessBilling (x14_No)
5	0.089165	x1_0	SeniorCitizen	SeniorCitizen (x1_0)
45	0.086919	x15_Mailed check	PaymentMethod	PaymentMethod (x15_Mailed check)
39	0.085343	x13_Two year	Contract	Contract (x13_Two year)
21	0.079725	x7_Yes	OnlineSecurity	OnlineSecurity (x7_Yes)
10	0.074368	x3_Yes	Dependents	Dependents (x3_Yes)
24	0.071112	x8_Yes	OnlineBackup	OnlineBackup (x8_Yes)
31	0.066610	x11_No	StreamingTV	StreamingTV (x11_No)
30	0.060710	x10_Yes	TechSupport	TechSupport (x10_Yes)
4	0.056564	x0_Male	gender	gender (x0_Male)
29	0.056017	x10_No internet service	TechSupport	TechSupport (x10_No internet service)

In [164]:
```
fig = plt.figure(figsize=(25, 10))
ax = plt.gca()

plt.bar(x = merged_svc['column_name_join'], height = merged_svc['0_x'])
plt.title('Linear SVC - Important Features (Magnitude of Coefficients)')
plt.xticks(rotation=90)
plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```