Problem Statement

Predicting housing prices is of interest to potential buyers, sellers, and organizations alike. Multiple online platforms offer, for example, a free "price estimate" based on underlying machine learning models. For this assignment, we are going to build the best machine learning model we can for Ames, Iowa. The data set consists of 79 features that describe the quality and quantity of the properties to base our predictions on.

Task 0: Data Preperation

*Note: No code has to be written for the 5 cells below - you may just execute them sequentially. After this, you may move on to **Task 1** on understanding the data.*

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score, precision_score, recall_score, acc
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.linear_model import Lasso, LassoCV, LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from scipy.stats import pearsonr
from scipy import stats
from sklearn import tree
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

In [2]:
```
##Filter warnings
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# All missing data removed/cleaned
housing_df = pd.read_csv("ames_data_no_missing.csv", index_col=0)
```

In [4]:
```
#Check the number of dummies to be created
count = [housing_df[col].nunique() for col in housing_df.columns if hous
sum(count)
```

Out[4]: 279

In [5]:
```
# ensure Python reads the categorical variables as categorical
for column in housing_df.columns:
    if housing_df[column].dtype == 'object':
        housing_df[column] = pd.Categorical(housing_df[column])
```

In [6]:
```
#define our RMSE function
def rmse(y_train, y_pred):
    return np.sqrt(mean_squared_error(y_train, y_pred))
```

Task 1: Understand the Data

Take some time to familiarize yourself with the data. It contains information about housing prices in Ames. What are the key variables?

You may perform any additional EDA if necessary.

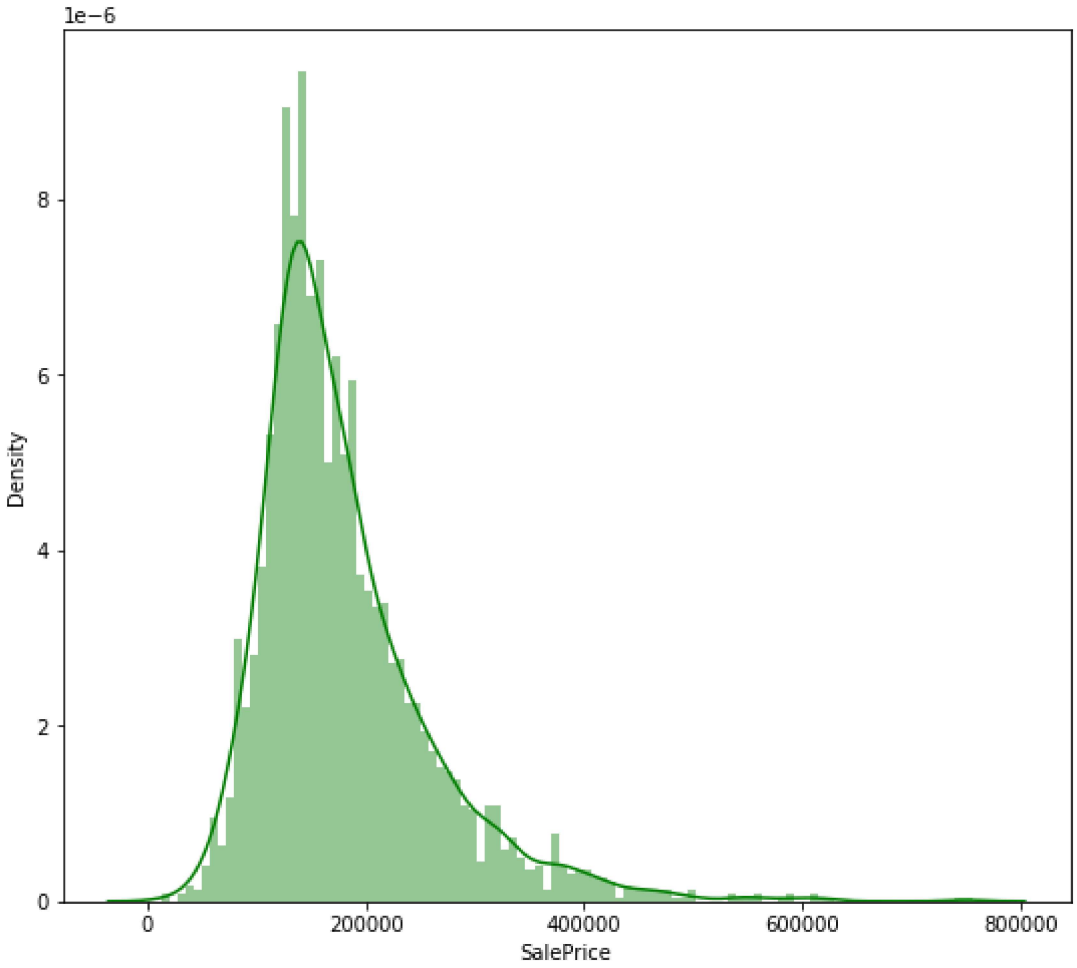
1.1

What is the distribution of housing prices?

In [7]:

```
# The original distribution

print(housing_df['SalePrice'].describe())          #distribution of hous
plt.figure(figsize=(9, 8))
sns.distplot(housing_df['SalePrice'], color='g', bins=100, hist_kws={'al
```

```
count      2930.000000
mean     180796.060068
std       79886.692357
min       12789.000000
25%      129500.000000
50%      160000.000000
75%      213500.000000
max      755000.000000
Name: SalePrice, dtype: float64
```



1.2

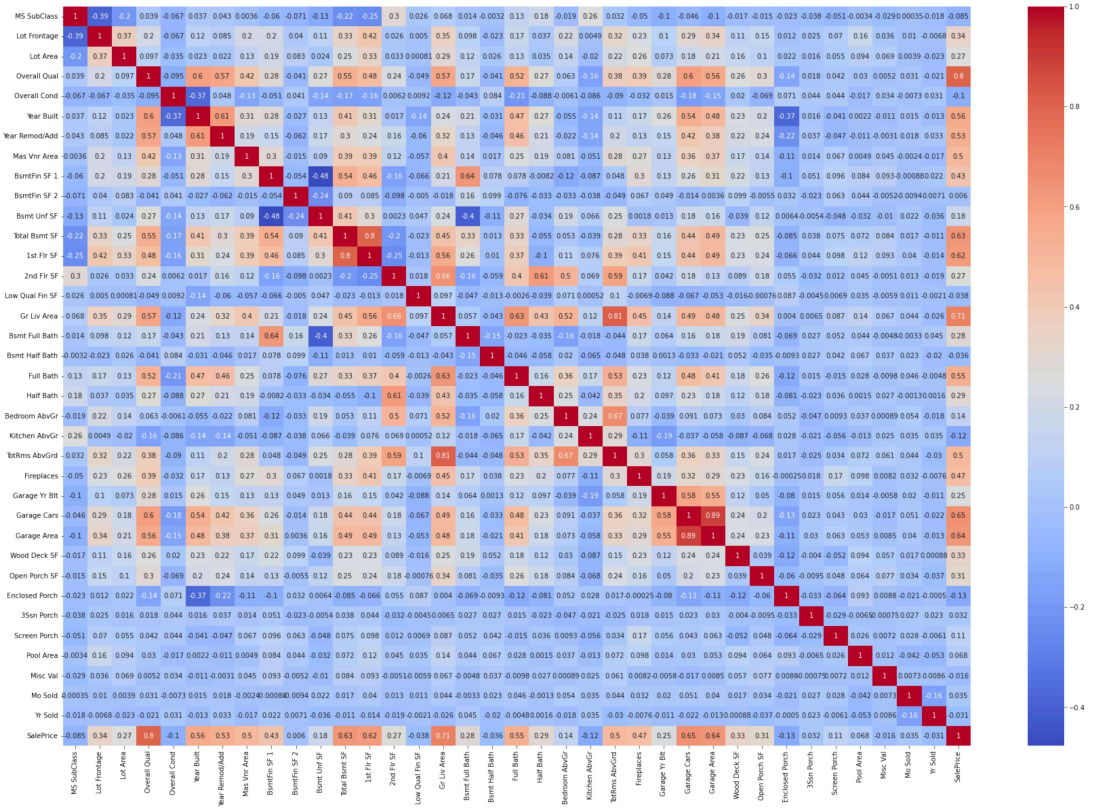
What is the variable that has the highest correlation with Housing prices? What are the key drivers behind larger house prices?

In [8]:
```
#Find the correlations of all variables with SalePrice

corr_df =housing_df.corr().sort_values(by='SalePrice',ascending=False).S
corr_df
```

Out[8]:
```
SalePrice          1.000000
Overall Qual       0.799262
Gr Liv Area        0.706780
Garage Cars        0.647562
Garage Area        0.640138
Total Bsmt SF      0.632529
1st Flr SF         0.621676
Year Built         0.558426
Full Bath          0.545604
Year Remod/Add     0.532974
Mas Vnr Area       0.502196
TotRms AbvGrd      0.495474
Fireplaces         0.474558
BsmtFin SF 1       0.433147
Lot Frontage       0.340484
Wood Deck SF       0.327143
Open Porch SF      0.312951
Half Bath          0.285056
Bsmt Full Bath     0.275823
2nd Flr SF         0.269373
Lot Area           0.266549
Garage Yr Blt      0.253459
Bsmt Unf SF        0.183308
Bedroom AbvGr      0.143913
Screen Porch       0.112151
Pool Area          0.068403
Mo Sold            0.035259
3Ssn Porch         0.032225
BsmtFin SF 2       0.006018
Misc Val          -0.015691
Yr Sold           -0.030569
Bsmt Half Bath    -0.035817
Low Qual Fin SF   -0.037660
MS SubClass       -0.085092
Overall Cond      -0.101697
Kitchen AbvGr     -0.119814
Enclosed Porch    -0.128787
Name: SalePrice, dtype: float64
```

In [9]:

```
plt.figure(figsize=(30,20))
sns.heatmap(housing_df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

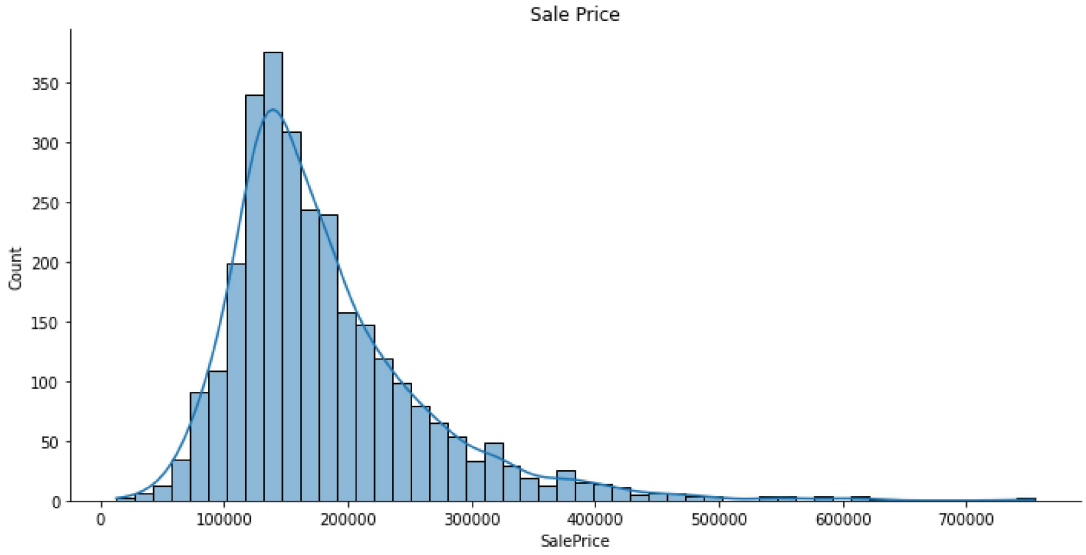


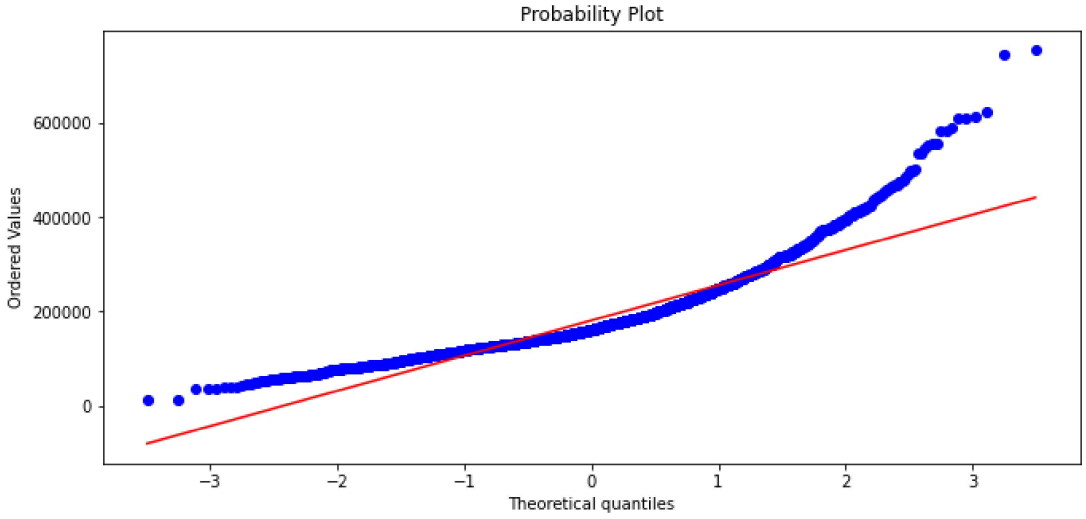
1.3

Create one additional visualization, that gives some insights into the data.

In [10]:

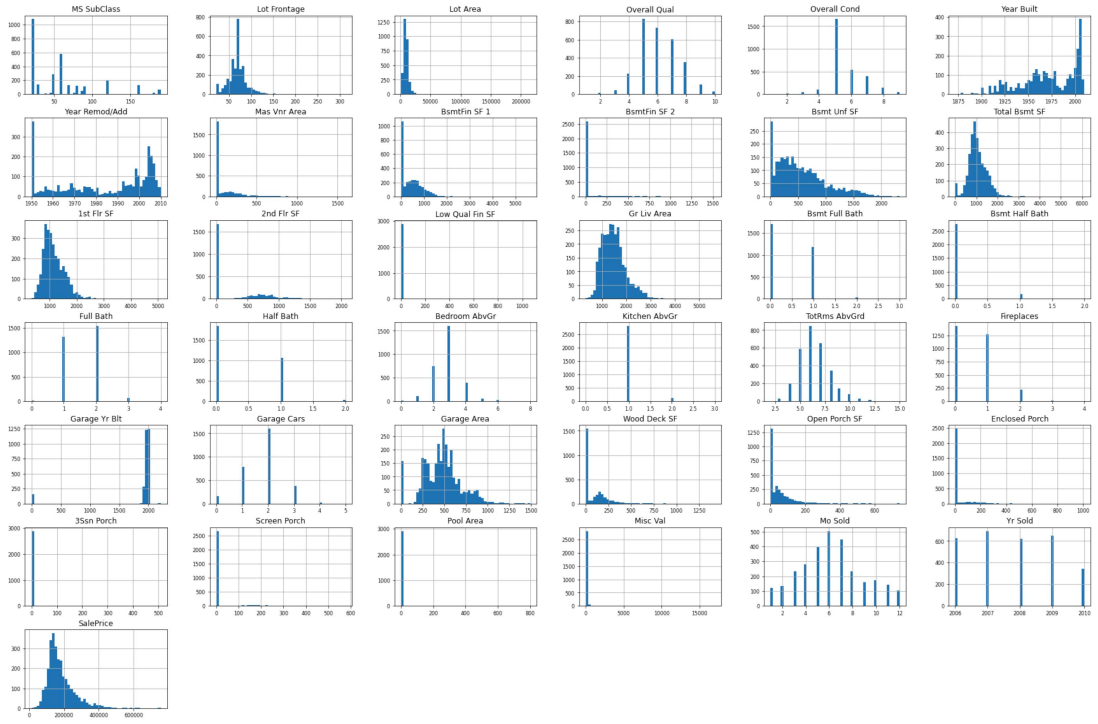
```
# a visualization to highlight any insight

sns.displot(data= housing_df, x=housing_df['SalePrice'],bins=50, kde=Tru
plt.title('Sale Price')
plt.show()

print('-'*250)

fig = plt.figure(figsize=(11,5))
res = stats.probplot(housing_df['SalePrice'], plot=plt)
plt.show()
```





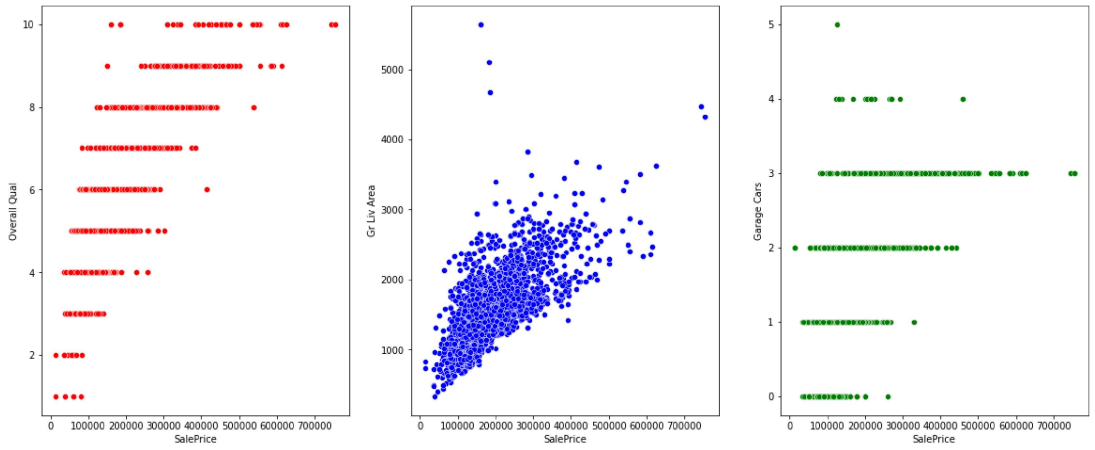
In [11]:
```
# a visualization to highlight any insight
df_hist =housing_df.select_dtypes(include=np.number)
df_hist.hist(figsize=(30, 20), bins=50, xlabelsize=8, ylabelsize=8)
```

Out[11]:
```
array([[<Axes: title={'center': 'MS SubClass'}>,
        <Axes: title={'center': 'Lot Frontage'}>,
        <Axes: title={'center': 'Lot Area'}>,
        <Axes: title={'center': 'Overall Qual'}>,
        <Axes: title={'center': 'Overall Cond'}>,
        <Axes: title={'center': 'Year Built'}>],
       [<Axes: title={'center': 'Year Remod/Add'}>,
        <Axes: title={'center': 'Mas Vnr Area'}>,
        <Axes: title={'center': 'BsmtFin SF 1'}>,
        <Axes: title={'center': 'BsmtFin SF 2'}>,
        <Axes: title={'center': 'Bsmt Unf SF'}>,
        <Axes: title={'center': 'Total Bsmt SF'}>],
       [<Axes: title={'center': '1st Flr SF'}>,
        <Axes: title={'center': '2nd Flr SF'}>,
        <Axes: title={'center': 'Low Qual Fin SF'}>,
        <Axes: title={'center': 'Gr Liv Area'}>,
        <Axes: title={'center': 'Bsmt Full Bath'}>,
        <Axes: title={'center': 'Bsmt Half Bath'}>],
       [<Axes: title={'center': 'Full Bath'}>,
        <Axes: title={'center': 'Half Bath'}>,
        <Axes: title={'center': 'Bedroom AbvGr'}>,
        <Axes: title={'center': 'Kitchen AbvGr'}>,
        <Axes: title={'center': 'TotRms AbvGrd'}>,
        <Axes: title={'center': 'Fireplaces'}>],
       [<Axes: title={'center': 'Garage Yr Blt'}>,
        <Axes: title={'center': 'Garage Cars'}>,
        <Axes: title={'center': 'Garage Area'}>,
        <Axes: title={'center': 'Wood Deck SF'}>,
        <Axes: title={'center': 'Open Porch SF'}>,
        <Axes: title={'center': 'Enclosed Porch'}>],
       [<Axes: title={'center': '3Ssn Porch'}>,
        <Axes: title={'center': 'Screen Porch'}>,
        <Axes: title={'center': 'Pool Area'}>,
        <Axes: title={'center': 'Misc Val'}>,
        <Axes: title={'center': 'Mo Sold'}>,
        <Axes: title={'center': 'Yr Sold'}>],
       [<Axes: title={'center': 'SalePrice'}>, <Axes: >, <Axes: >,
        <Axes: >, <Axes: >, <Axes: >]], dtype=object)
```



In [12]:

```
# Scatter from top 3 correlaltion
plt.subplots(figsize=(20,8))

plt.subplot(1,3,1)
sns.scatterplot(data=housing_df, x=housing_df['SalePrice'], y=housing_df

plt.subplot(1,3,2)
sns.scatterplot(data=housing_df, x=housing_df['SalePrice'], y=housing_df

plt.subplot(1,3,3)
sns.scatterplot(data=housing_df, x=housing_df['SalePrice'], y=housing_df

plt.show()
```



Task 2: Build machine learning models

Use your knowledge of prediction models to create at least three models that predict housing prices.

2.1

1. *Create dummies for all the categorical columns.*
2. *Partition your data into training and validation (70-30 split, setting the random state to 1).*
3. *Scale the train and the test set using StandardScaler()*

In [13]:
```
# Initialize X and y
X = housing_df.drop(columns=['SalePrice']) # All but the outcome column
y = housing_df['SalePrice']
```

In [14]:
```
# Use dummy variables for categorical variables

X = pd.get_dummies(X, drop_first=False) #for kNN and trees
X2 = pd.get_dummies(X, drop_first=True) #for regression
```

In [15]:
```
# Train - Test split (70-30 split, setting the random state to 1)

X_train, X_val, X2_train, X2_val, y_train, y_val = train_test_split(X, )
```

In [16]:
```
# Scale the train and test set features separately
scaler = StandardScaler()
numeric_cols = [col for col in X.columns if X[col].dtypes != 'category']

X_train[numeric_cols]= scaler.fit_transform(X_train[numeric_cols])
X_val[numeric_cols]= scaler.transform(X_val[numeric_cols])
```

In [17]:
```
X_scaled = scaler.fit_transform(X)
Xsc_train, Xsc_val, y_train, y_val = train_test_split(X_scaled, y, test_
```

2.2

Build a linear regression model, a regression tree and a kNN model. Carefully apply regularization for the linear regression model. Carefully select which variables to use for the kNN model.

In [18]:
```
# Linear model - USE LassoCV to get the best LASSO model

from sklearn.linear_model import Lasso, LassoCV, Ridge, RidgeCV

lasso_model = Lasso(alpha=1)
lasso_model.fit(Xsc_train, y_train)
```

Out[18]:
```
Lasso(alpha=1)
```

In [19]:
```
print('RMSE training set', round(rmse(y_train, lasso_model.predict(Xsc_t
print('RMSE validation set', round(rmse(y_val, lasso_model.predict(Xsc_v
```

```
RMSE training set 20652.6
RMSE validation set 22663.0
```

In [20]:
```
coef_table = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(np.transpos
coef_table.columns = ['', 'Coefficient']
coef_table.set_index('', inplace=True)
coef_table
```

Out[20]:

	Coefficient
MS SubClass	-2631.531173
Lot Frontage	-660.895223
Lot Area	4511.909531
Overall Qual	9942.762625
Overall Cond	6430.863272
...	...
Sale Condition_AdjLand	577.480201
Sale Condition_Alloca	591.857915
Sale Condition_Family	-1433.892834
Sale Condition_Normal	1255.123061
Sale Condition_Partial	-434.357848

315 rows × 1 columns

In [21]:
```
alphas = np.arange(0.01, 50, .5) # Check all alphas from .01 to 10 in st
lasso_cv_model = LassoCV(alphas= alphas , cv=5, max_iter=50000)
# fit model
lasso_cv_model.fit(Xsc_train, y_train)
# summarize chosen configuration
print('alpha: %f' % lasso_cv_model.alpha_)
```

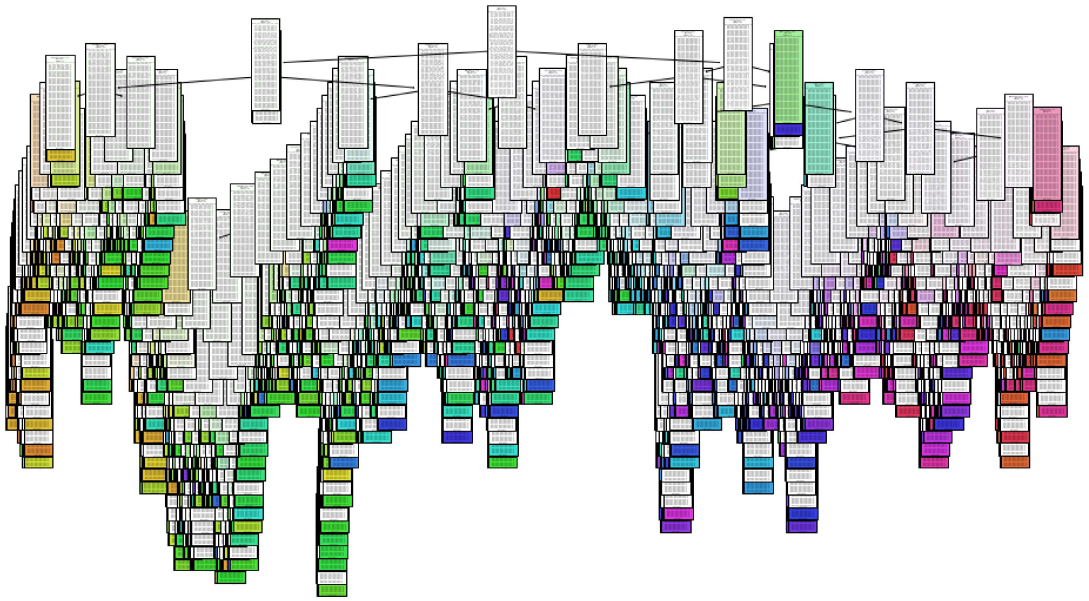
```
alpha: 23.010000
```

In [22]:
```
# Set best alpha
lasso_best_model = Lasso(alpha=lasso_cv_model.alpha_)
lasso_best_model.fit(Xsc_train, y_train)

print('Best Lasso-RMSE training set', round(rmse(y_train, lasso_best_mod
print('Best Lasso-RMSE validation set', round(rmse(y_val, lasso_best_mod
```

```
Best Lasso-RMSE training set 20671.35
Best Lasso-RMSE validation set 22318.498
```

In [23]:
```
# Tree Model - Use max depth to control the complexity of the tree. Run

tree_model_1 = DecisionTreeClassifier(random_state = 1)
tree_model_1 = tree_model_1.fit(X_train, y_train)
```

In [24]:
```
# Visualize the decision tree for 'tree_model_1'
# Note: This cell may take a while to run owing to the large number of r

fig = plt.figure(figsize = (16,8))
fig = tree.plot_tree(tree_model_1, feature_names = X.columns, filled = T
```



In [25]:
```
# Print the number of leaves and the depth of the tree for 'tree_model_1
tree_model_1_n_leaves = tree_model_1.get_n_leaves()
tree_model_1_depth = tree_model_1.get_depth()
print('Number of leaves =', tree_model_1_n_leaves)
print('Tree depth =', tree_model_1_depth)
```

```
Number of leaves = 1774
Tree depth = 39
```

In [26]:
```
# Obtain predicted class labels for the training and validation data usi
# Hint: Study the documentation of the 'predict()' method
y_pred_1_train = tree_model_1.predict(X_train)
y_pred_1_val = tree_model_1.predict(X_val)
```

In [27]:
```
# Compute the accuracy and the sensitivity of 'tree_model_1' on the trai
# Note: The 'pos_label' parameter for the 'recall_score()' method should
# Note: The positive class label in this exercise is '1', which is also
acc_train_1 = accuracy_score(y_train, y_pred_1_train)
acc_val_1 = accuracy_score(y_val, y_pred_1_val)

# Summarize the above metrics for the train and validation sets using a
tree_model_1_metrics = pd.DataFrame(data = {'Accuracy': [acc_train_1, ac
                                            },
                                    index = ['tree_model_1_train', 'tree

tree_model_1_metrics
```

Out[27]:

	Accuracy
tree_model_1_train	0.999512
tree_model_1_val	0.013652

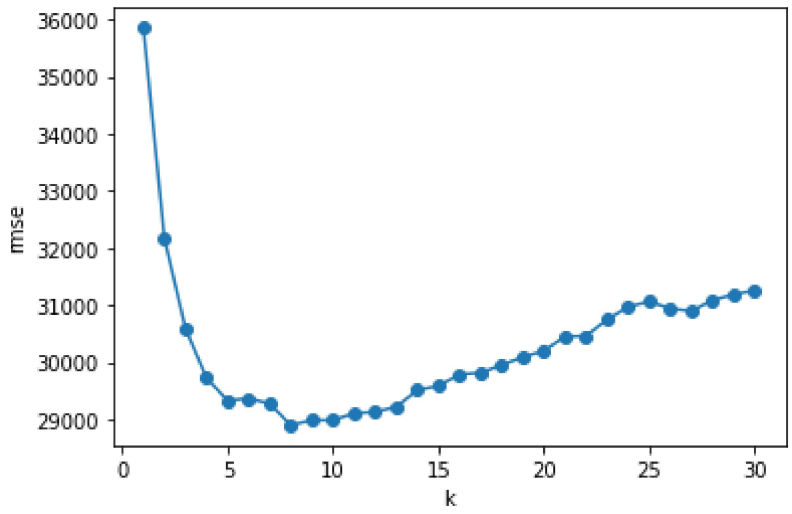
In [28]:
```
y = housing_df['SalePrice']
y_train, y_val = train_test_split(y, test_size=0.3, random_state = 1)
```

In [29]:
```
#rmse function
def rmse(y_train, y_pred):
  return np.sqrt(mean_squared_error(y_train, y_pred))
```

In [30]:

```
# KNN Model

# Select the top 20 most correlated features and store it in a list call

top_20_features= list(corr_df.head(21).index)
top_20_features.remove('SalePrice')

#For building the model, you must use X_train[top_20_features]

##### CODE HERE #####
X_train[top_20_features]= scaler.fit_transform(X_train[top_20_features])
X_val[top_20_features]= scaler.transform(X_val[top_20_features])



# Find the value of k for which RMSE is minimum, using GridSearchCV

##### CODE HERE #####
kvalues = np.arange(1,31) # Parameter range

val_rmse=[]

for k in kvalues:
  knn_reg = KNeighborsRegressor(n_neighbors=k)
  knn_reg.fit(X_train[top_20_features], y_train)
  y_pred = knn_reg.predict(X_val[top_20_features])
  val_rmse.append(rmse(y_val, y_pred))

# Find the value of k for which RMSE is minimum, using GridSearchCV

##### CODE HERE #####
### Rmse vs k
plt.plot(kvalues,val_rmse,marker='o')
plt.xlabel("k")
plt.ylabel("rmse")
print("The minimum rmse is obtained at k = " + str(np.argmin(val_rmse)+1
```

```
The minimum rmse is obtained at k = 8
```



In [31]:
```
knn_reg_best = KNeighborsClassifier(n_neighbors=29)
knn_reg_best.fit(X_train[top_20_features], y_train)
print('RMSE validation set:', round(rmse(y_val, knn_reg_best.predict(X_v
```

```
RMSE validation set: 49394.3
```

2.3

Summarize the predictive performance in terms of RMSE.

1. *Calculate the RMSE values for train and validation for all the models*
2. *Display them in a tabulated format*

Hint: You may use the code that you've learnt in the 'Model selection' module

In [32]:

```
#Linear regression

##### CODE HERE #####
alphas = np.arange(.01, 25, .25) # Check all alphas from .01 to 10 in st
lasso_cv_model = LassoCV(alphas= alphas, cv=5, max_iter=50000)
lasso_cv_model.fit(X2_train, y_train)

lin_reg_best = Lasso(alpha=lasso_cv_model.alpha_)
lin_reg_best.fit(X2_train, y_train)

lin_train_rmse = rmse(y_train, lin_reg_best.predict(X2_train))
lin_val_rmse = rmse(y_val, lin_reg_best.predict(X2_val))



#max depth pruned tree

##### CODE HERE #####
tree_reg = DecisionTreeRegressor(max_depth=5, random_state=0)
path= tree_reg.cost_complexity_pruning_path(X_train,y_train)
ccp_alphas = path.ccp_alphas
#print(ccp_alphas)

#fit the trees
regs=[]

for ccp_alpha in ccp_alphas:
  #train
  curr_reg = DecisionTreeRegressor(random_state=0, ccp_alpha = ccp_alpha
  curr_reg.fit(X_train,y_train)

  #save the model
  regs.append(curr_reg)

#calculate the rmse for all the tree
train_rmse=[]
val_rmse=[]

#loop through models
for r in regs:
  y_train_pred=r.predict(X_train)
  y_val_pred = r.predict(X_val)

  train_rmse.append(rmse(y_train_pred,y_train))
  val_rmse.append(rmse(y_val_pred,y_val))

#now pick the best one
best_ccp_alpha = ccp_alphas[val_rmse.index(min(val_rmse))]
print('Best CCP aplha',best_ccp_alpha)

# train the corresponding tree
tree_reg_best = tree.DecisionTreeRegressor(random_state=0, ccp_alpha=bes
tree_reg_best.fit(X_train,y_train)

tree_train_rmse = rmse(y_train, tree_reg_best.predict(X_train))
tree_val_rmse = rmse(y_val, tree_reg_best.predict(X_val))



#knn
```

```
##### CODE HERE #####
knn_reg_best = KNeighborsRegressor(n_neighbors=16)
knn_reg_best.fit(X_train, y_train)

knn_train_rmse = rmse(y_train, knn_reg_best.predict(X_train))
knn_val_rmse = rmse(y_val, knn_reg_best.predict(X_val))




#Display the RMSEs

##### CODE HERE #####
pd.DataFrame([[lin_train_rmse, lin_val_rmse],[tree_train_rmse, tree_val_
              [knn_train_rmse, knn_val_rmse]], columns=['RMSE Train', 'R
              index = ['Linear', 'Tree', 'kNN'])
```

```
Best CCP aplha 17852317.90450956
```

Out[32]:

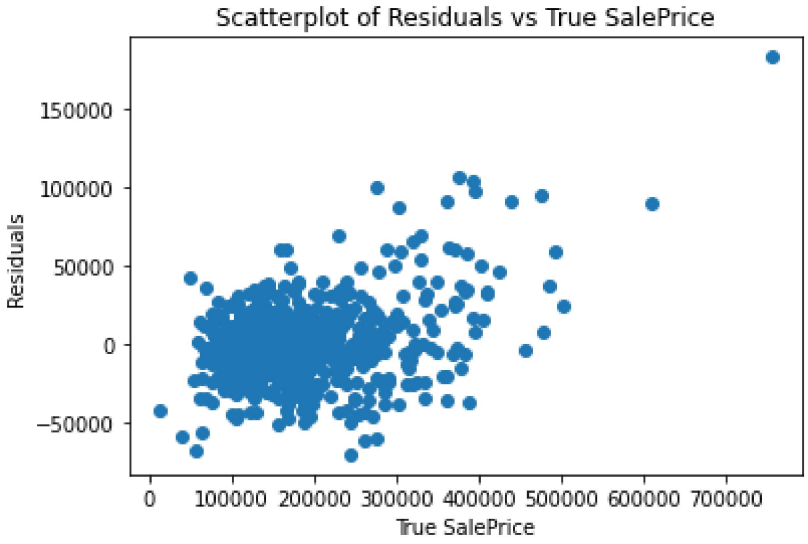
	RMSE Train	RMSE Validation
Linear	21369.268396	22593.917397
Tree	29017.245304	32841.378545
kNN	35785.901409	37633.611024

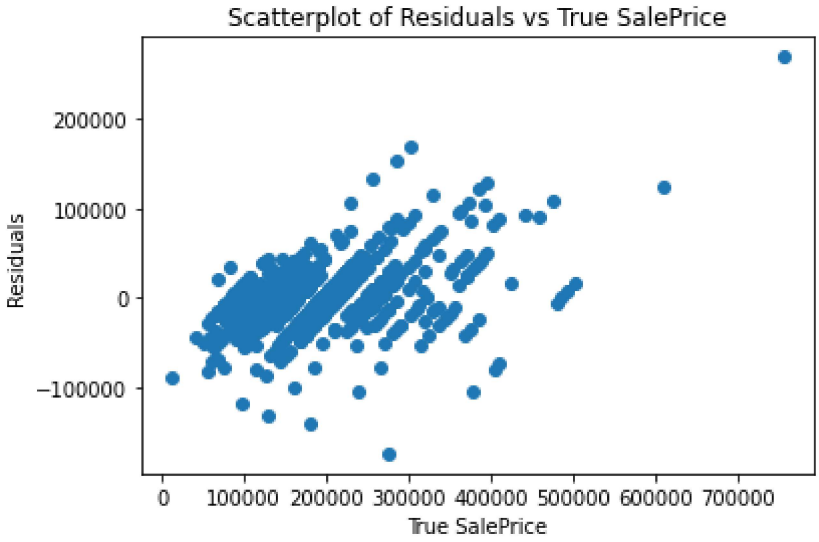
2.4

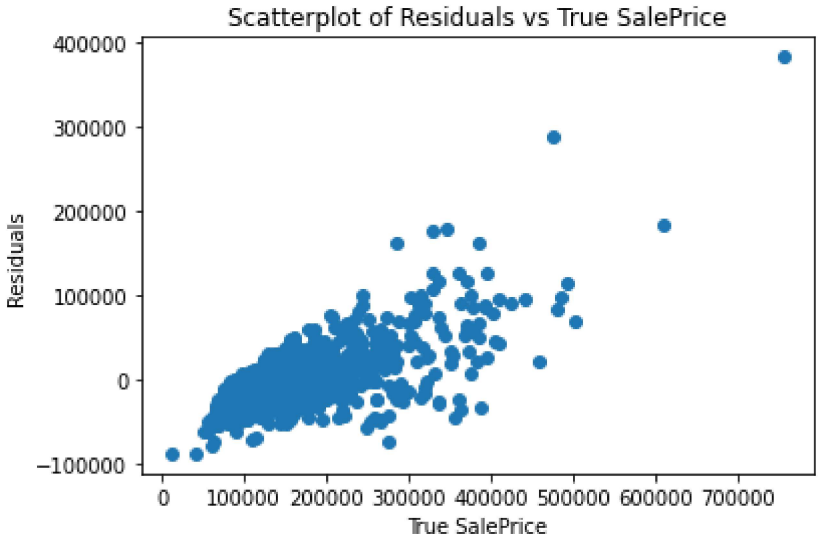
Study the largest errors that you made (largest overpredictions, largest underpredictions). What may be some of the reasons why the model is over/under predicting? Do these insights possibly help you improve the models?

In [33]:

```
# Visualize the errors - plot a scatterplot of the residuals vs the true

lasso_best = Lasso(alpha = lasso_cv_model.alpha_)
lasso_best.fit(Xsc_train, y_train)

y_val_pred=lasso_best.predict(Xsc_val)

residuals_lasso = y_val - y_val_pred

tree_reg = DecisionTreeRegressor(max_depth=5, random_state = 0)
tree_reg = tree_reg.fit(X_train, y_train)

y_val_pred=tree_reg.predict(Xsc_val)

residuals_tree = y_val - y_val_pred

knn = KNeighborsRegressor(n_neighbors=8)
knn.fit(X_train, y_train)

y_val_pred=knn.predict(Xsc_val)

residuals_knn = y_val - y_val_pred

plt.scatter(y_val, residuals_lasso)
plt.xlabel("True SalePrice")
plt.ylabel("Residuals")
plt.title("Scatterplot of Residuals vs True SalePrice")
plt.show()

plt.scatter(y_val, residuals_tree)
plt.xlabel("True SalePrice")
plt.ylabel("Residuals")
plt.title("Scatterplot of Residuals vs True SalePrice")
plt.show()

plt.scatter(y_val, residuals_knn)
plt.xlabel("True SalePrice")
plt.ylabel("Residuals")
plt.title("Scatterplot of Residuals vs True SalePrice")
plt.show()
```







Task 3

3.1

Are you able to improve your linear regression model by taking the log of the dependent variable? (remember to translate your predicted outcome back to the original units before calculating the RMSE)

Create a visualization, that highlights the distribution of prices when after taking log of the dependent variable

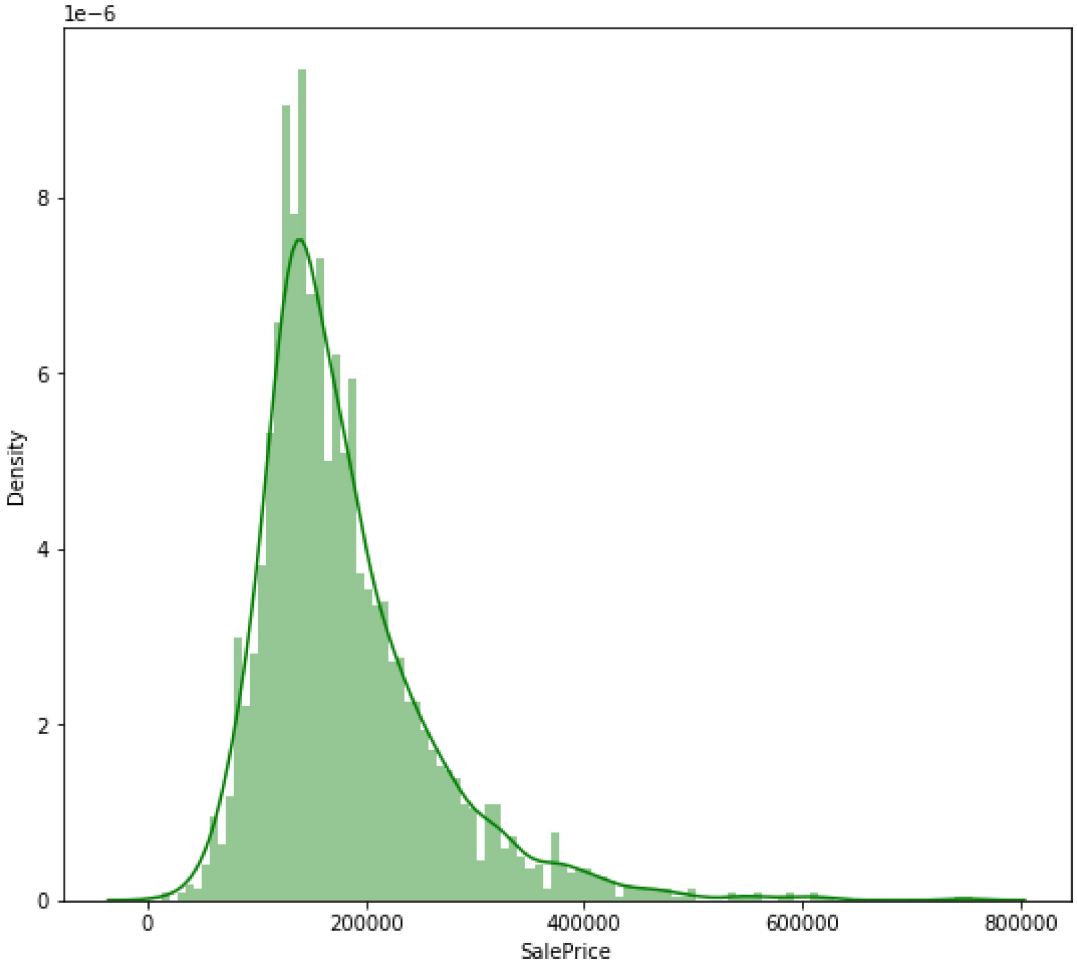
Hint - You may use numpy.log() (https://numpy.org/doc/stable/reference/generated/numpy.log.html) to get the log of the dependent variable

In [34]:
```
y = housing_df['SalePrice']
y_train, y_val = train_test_split(y, test_size=0.3, random_state = 1)
```

In [35]:

```
# distribution of the transformed SalePrice


print(housing_df['SalePrice'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(housing_df['SalePrice'], color='g', bins=100, hist_kws={'al
```

```
count      2930.000000
mean     180796.060068
std       79886.692357
min       12789.000000
25%      129500.000000
50%      160000.000000
75%      213500.000000
max      755000.000000
Name: SalePrice, dtype: float64
```



In [36]:
```
# Linear model - Using the log of the SalePrice as the dependent variabl
# Note that the optimum value of Alpha for this model would also be scal


# Log Model
log_clf_best = LogisticRegression(penalty='none', solver='lbfgs', random
                                  max_iter=200).fit(X2_train, y_train
log_train_acc = log_clf_best.score(X2_train, y_train)
log_val_acc = log_clf_best.score(X2_val, y_val)

# Tree Model
best_ccp_alpha = 0.004801587301587302
tree_clf_best = DecisionTreeClassifier(random_state=0, ccp_alpha=best_cc
tree_clf_best.fit(Xsc_train,y_train)

tree_train_acc = tree_clf_best.score(Xsc_train, y_train)
tree_val_acc = tree_clf_best.score(Xsc_val, y_val)

# KNN Model
knn_clf_best = KNeighborsClassifier(n_neighbors=14)
knn_clf_best.fit(Xsc_train, y_train)

knn_train_acc = knn_clf_best.score(Xsc_train, y_train)
knn_val_acc = knn_clf_best.score(Xsc_val, y_val)
```

In [37]:
```
# Calculate the RMSE values for train and the test set
y_pred_train=lasso_best.predict(X2_train)
y_pred_val=lasso_best.predict(X2_val)

#tranforming the prediction back to the original units
y_pred_train=np.exp(y_train)
y_pred_val=np.exp(y_val_pred)

#calculating RMSE for training and validation datasets
lr_train_rmse=rmse(y_train,y_train)
lr_val_rmse=rmse(y_val,y_val_pred)

print('RMSE value for training dataset is', lr_train_rmse)
print('RMSE value for validation dataset is', lr_val_rmse)
```

```
RMSE value for training dataset is 0.0
RMSE value for validation dataset is 36778.87216263759
```

In [38]:
```
# Display the RMSE values in a dataframe

pd.DataFrame([[lin_train_rmse, lin_val_rmse],[tree_train_rmse, tree_val_
              [knn_train_rmse, knn_val_rmse]], columns=['RMSE Train', 'R
              index = ['Linear', 'Tree', 'kNN'])
```

Out[38]:

	RMSE Train	RMSE Validation
Linear	21369.268396	22593.917397
Tree	29017.245304	32841.378545
kNN	35785.901409	37633.611024

3.2 Bonus Task

Experiment with data segmentation: Should you subset the data and fit separate models for each subset?

Data segmentation is generally useful when we think that subsegments of our data have substantially different relationships between their features and the outcome compared to other subsegments (i.e variable interactions). We can use a combination of prior knowledge and data exploration to build our domain knowledge about where this situation would apply.

Starting with prior knowledge, you can hypothesize $HouseStyle$ may be a candidate for data segmentation, as for instance, 3 bedrooms in a 1-story house may have a different effect on $SalePrice$ than 3 bedrooms in a 2-story house.

In [39]:
```
housing_df['House Style'].value_counts()
```

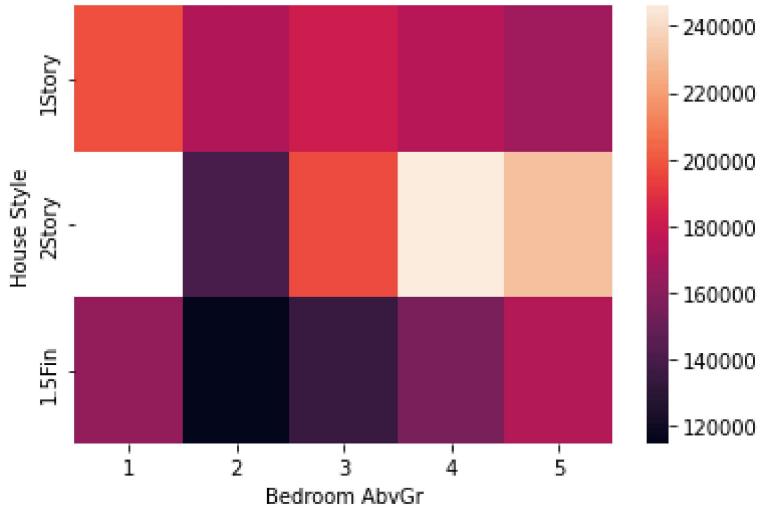
Out[39]:
```
1Story    1481
2Story     873
1.5Fin     314
SLvl       128
SFoyer      83
2.5Unf      24
1.5Unf      19
2.5Fin       8
Name: House Style, dtype: int64
```

In [40]:
```
housing_df['Bedroom AbvGr'].value_counts()
```

Out[40]:
```
3    1597
2     743
4     400
1     112
5      48
6      21
0       8
8       1
Name: Bedroom AbvGr, dtype: int64
```

In [41]:

```
matrix = []
styles = ['1Story', '2Story', '1.5Fin']
for style in styles:
    curr_style = []
    for bedrooms in range(1, 6):
        curr_mean = housing_df[(housing_df['House Style'] == style) &
                               (housing_df['Bedroom AbvGr'] == bedrooms)

        curr_style.append(curr_mean)
    matrix.append(curr_style)
sns.heatmap(matrix)
plt.ylabel('House Style')
plt.yticks(np.arange(3)+0.5, styles)
plt.xlabel('Bedroom AbvGr')
plt.xticks(np.arange(5)+0.5, np.arange(5)+1)
pass
```



We indeed see some interaction between the housing style and bedroom number, indicating data segmentation could be promising.

From here, it's your task to start building a linear model to see whether data segmentation will improve results.

Hint: For the first two subtasks in 3.2, you could run a for-loop for each style in HouseStyles and evaluate/create the LASSO model.

In [42]:

```
# Linear Full Model (FM) - Train a Lasso model for the whole dataset

subset_data=housing_df[housing_df['House Style'].isin(styles)]

X=subset_data.iloc[:,:-1]
y=subset_data['SalePrice']

#standardizing
scaler=StandardScaler()
numeric_cols=[col for col in X.columns if X[col].dtypes!='category']
X[numeric_cols]=scaler.fit_transform(X[numeric_cols])

X=pd.get_dummies(X,drop_first=True)

#splitting data
X_train,X_val,y_train,y_val=train_test_split(X,y,test_size=0.3,random_st

#fitting  lasso model for whole subset
model_whole=LassoCV(alphas=np.arange (0.01,25,0.25),cv=5,max_iter=50000)
model_whole.fit(X_train,y_train)

#training lasso regression modelusing best alpha
model_whole_best=Lasso(alpha=model_whole.alpha_)
model_whole_best.fit(X_train, y_train)

#calculating best RMSE values for training & validation datasets
rmse_whole_train=[]
rmse_subset_train=[]
rmse_whole_val=[]
rmse_subset_val=[]

for style in styles:
    #seperating subsets
    subset_X_train=X_train[subset_data['House Style']==style]
    subset_y_train=y_train[subset_data['House Style']==style]
    subset_X_val=X_val[subset_data['House Style']==style]
    subset_y_val=y_val[subset_data['House Style']==style]

   #storing RMSE value for 3 subsets of styles
    RMSE_train=rmse(subset_y_train,model_whole_best.predict(subset_X_tra
    rmse_whole_train.append(RMSE_train)
    RMSE_val=rmse(subset_y_val,model_whole_best.predict(subset_X_val))
    rmse_whole_val.append(RMSE_val)

   #using cross validation to select optimal regularization parameter
    model=LassoCV(alphas=np.arange(0.01,25,0.25),cv=5,max_iter=50000)
    model.fit(subset_X_train, subset_y_train)

   #training Lasso regression model for different styles using best alph
    model_best=Lasso(alpha=model.alpha_)
    model_best.fit(subset_X_train,subset_y_train)

   #storing RMSE value for 3 subset of styles
    RMSE_subset_train= rmse(subset_y_train, model_best.predict(subset_X_
    rmse_subset_train.append(RMSE_subset_train)
    RMSE_subset_val=rmse(subset_y_val, model_best.predict(subset_X_val))
    rmse_subset_val.append(RMSE_subset_val)
```

In [43]:
```
# Create a DataFrame to store the values of RMSE for both the models on

rmse_DataFrame=pd.DataFrame([rmse_whole_train,rmse_whole_val,rmse_subset
round(rmse_DataFrame,2)
```

Out[43]:

	1Story	2Story	1.5Fin
whole_train	21729.79	22502.10	16537.86
whole_val	27183.31	51648.18	19660.65
subset_train	19799.56	15619.44	7560.99
subset_val	24779.24	66275.60	62481.72

As we can clearly see from the table that the RMSE values for most the styles have decreased significantly after subsetting. so in this case subsetting might be a good idea.

In [44]:

```
# Linear Data Segmentation Model (DSM) - Train a Lasso model for the ind

##### CODE HERE #####
# Define the predictors and the outcome variable
predictors = ['Bedroom AbvGr']
outcome = 'SalePrice'

# Define the combinations of house style and number of bedrooms to segme
combinations = [('1Story', 3)]


# Loop through each combination of house style and number of bedrooms ar
for style, bedrooms in combinations:
# Segment the data by house style and number of bedrooms
    key = style + '_' + str(bedrooms)
    subset_data = housing_df[(housing_df['House Style'] == style) & (hou

# Split the data into training and validation sets
X = subset_data[predictors]
y = subset_data[outcome]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, r

# Fit a LASSO model
model = Lasso(alpha=0.1)
model.fit(X_train, y_train)

# Compute the RMSE for the training and validation sets
y_train_pred = model.predict(X_train)
y_val_pred = model.predict(X_val)
train_rmse_c = round(mean_squared_error(y_train, y_train_pred, squared=F
val_rmse_c = round(mean_squared_error(y_val, y_val_pred, squared=False))



# Define the predictors and the outcome variable
predictors = ['Bedroom AbvGr']
outcome = 'SalePrice'

# Define the combinations of house style and number of bedrooms to segme
combinations1 = [('2Story', 3)]


# Loop through each combination of house style and number of bedrooms ar
for style, bedrooms in combinations1:
# Segment the data by house style and number of bedrooms
    key = style + '_' + str(bedrooms)
    subset_data = housing_df[(housing_df['House Style'] == style) & (hou

# Split the data into training and validation sets
X = subset_data[predictors]
y = subset_data[outcome]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, r

# Fit a LASSO model
model = Lasso(alpha=0.1)
model.fit(X_train, y_train)

# Compute the RMSE for the training and validation sets
y_train_pred = model.predict(X_train)
y_val_pred = model.predict(X_val)
train_rmse_c1 = round(mean_squared_error(y_train, y_train_pred, squared=
```

```
val_rmse_c1 = round(mean_squared_error(y_val, y_val_pred, squared=False)



# Define the predictors and the outcome variable
predictors = ['Bedroom AbvGr']
outcome = 'SalePrice'

# Define the combinations of house style and number of bedrooms to segme
combinations2 = [('1.5Fin', 3)]


# Loop through each combination of house style and number of bedrooms ar
for style, bedrooms in combinations2:
# Segment the data by house style and number of bedrooms
    key = style + '_' + str(bedrooms)
    subset_data = housing_df[(housing_df['House Style'] == style) & (hou

# Split the data into training and validation sets
X = subset_data[predictors]
y = subset_data[outcome]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, r

# Fit a LASSO model
model = Lasso(alpha=0.1)
model.fit(X_train, y_train)

# Compute the RMSE for the training and validation sets
y_train_pred = model.predict(X_train)
y_val_pred = model.predict(X_val)
train_rmse_c2 = round(mean_squared_error(y_train, y_train_pred, squared=
val_rmse_c2 = round(mean_squared_error(y_val, y_val_pred, squared=False)


# Store the RMSE values of train and validation for all the 3 subsets of

##### CODE HERE #####
pd.DataFrame([[train_rmse_c, val_rmse_c],[train_rmse_c1, val_rmse_c1],[t
```

Out[44]:

	Train_RMSE	Validation_RMSE
1Story	66474	80585
2Story	66688	61662
1.5Fin	32839	41194

In [45]:
```
# Create a DataFrame to store the values of RMSE for both the models on

##### CODE HERE #####
pd.DataFrame([[train_rmse_c, val_rmse_c],[train_rmse_c1, val_rmse_c1],[t
```

Out[45]:

	Train_RMSE	Validation_RMSE
1Story	66474	80585
2Story	66688	61662
1.5Fin	32839	41194

Write down your inferences about the performance of the subsetted model here -

The subsettled models trained on three different house styles show varying performance on both the training and validation data. The validation and training RMSE for all three models indicating that they maybe overfitting the training data. Additionally, the predictive power of the models lowest the 2 story style having the lowest validation RMSE and the 1 Story style having the highest. Lastly the difference between training and validation models which may be due to factors such as the sample size or quality of the data. These findings suggest that while the models provide some house style and sale price, further analysis and validation may be necessary to draw firm conclusions.

Task 4: Summarize your findings

Now take some time to translate your results into valuable insights.

4.1

What drives housing prices? Find the top 20 major drivers.

Hint - In course 3 module 1, you have already seen how to store the coefficients of a model in a dictionary. You can convert the dictionary into a DataFrame and sort the dataframe by the coefficients. [Here's (https://stackoverflow.com/questions/18837262/convert-python-dict-into-a-dataframe)](https://stackoverflow.com/questions/18837262/convert-python-dict-into-a-dataframe) some guidance on how to convert dictionary into a DataFrame.

In [46]:

```
# Visualize all the columns and their coefficients sorted in descending
# Hint - Check the code for Course 3 Module 1 - Linear regression in a p

##### CODE HERE #####
X=housing_df[['Overall Qual','Gr Liv Area', 'Garage Cars', 'Garage Area'
              'Full Bath', 'Year Remod/Add', 'Mas Vnr Area', 'TotRms Abv
              'Lot Frontage', 'Wood Deck SF', 'Open Porch SF', 'Half Bat
y=housing_df['SalePrice']
#Train a lasso model
lasso=Lasso(alpha=0.1)
lasso.fit(X,y)

#store the coefficients in a dictionary
coef_dict={'Feature':X.columns,'coefficient':lasso.coef_}

#convert the dictionary to dataframe
coef_df=pd.DataFrame.from_dict(coef_dict)

#sort the dataframe by the coefficiant values
coef_df = round(coef_df.sort_values('coefficient',ascending=False))
coef_df
```

Out[46]:

	Feature	coefficient
0	Overall Qual	19332.0
11	Fireplaces	6356.0
17	Bsmt Full Bath	6289.0
2	Garage Cars	5065.0
8	Year Remod/Add	370.0
6	Year Built	197.0
13	Lot Frontage	103.0
9	Mas Vnr Area	33.0
5	1st Flr SF	26.0
1	Gr Liv Area	25.0
3	Garage Area	22.0
18	2nd Flr SF	21.0
14	Wood Deck SF	18.0
12	BsmtFin SF 1	14.0
4	Total Bsmt SF	12.0
19	Lot Area	0.0
15	Open Porch SF	-7.0
10	TotRms AbvGrd	-387.0
16	Half Bath	-1867.0
7	Full Bath	-2478.0

You can also use a built in variable importance function from decision trees to capture a summary of the importance of different features in our regression tree.

Note: There is no coding to be done in this cell. Just execute this cell and observe the

In [47]:
```
# Extract the feature_importances_ attribute from the tree model (featur

# Extracting the importances by sklearn (Replace tree_reg_best by the va
importances_sk = tree_reg_best.feature_importances_

# Creating a dataframe with the feature importance by sklearn
feature_importance_df = []
for i, feature in enumerate(X_train.columns):
    feature_importance_df.append([feature, round(importances_sk[i], 3)])

feature_importance_df = pd.DataFrame(feature_importance_df,
                                     columns=['Feature', 'Importance'])
feature_importance_df = feature_importance_df.sort_values(by='Importance
print(f"Feature importance by sklearn: ")
feature_importance_df.iloc[:20]
```

```
Feature importance by sklearn:
```

Out[47]:

	Feature	Importance
0	Bedroom AbvGr	0.0

4.2

What is the predictive performance of your models?

In [48]:
```
# Compare the RMSE of the train and the validation set for all the model
pd.DataFrame([[lr_train_rmse,lr_val_rmse],[tree_train_rmse,tree_val_rmse
```

Out[48]:

	RMSE Train	RMSE Validation
Linear	0.000000	36778.872163
Tree	29017.245304	32841.378545
kNN	35785.901409	37633.611024

Which model performs the best?

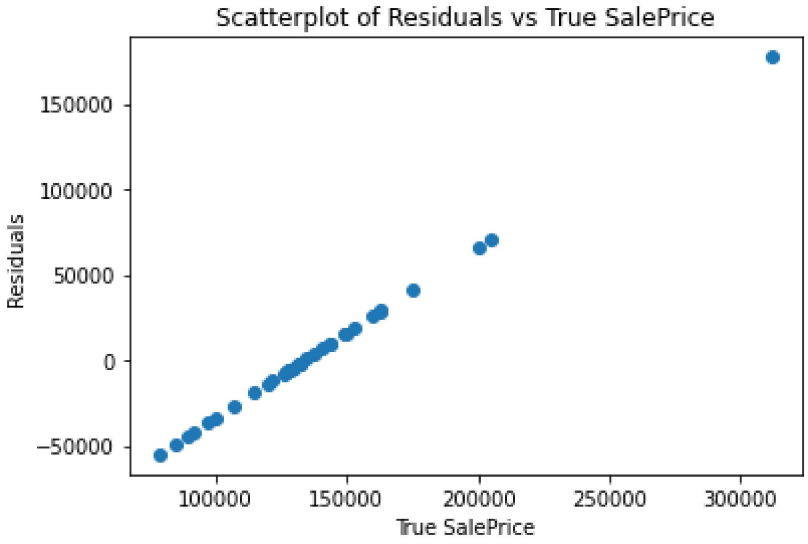
From comparing all three models by their RMSE score in validation set,we can clearly see that Tree regression model work best.

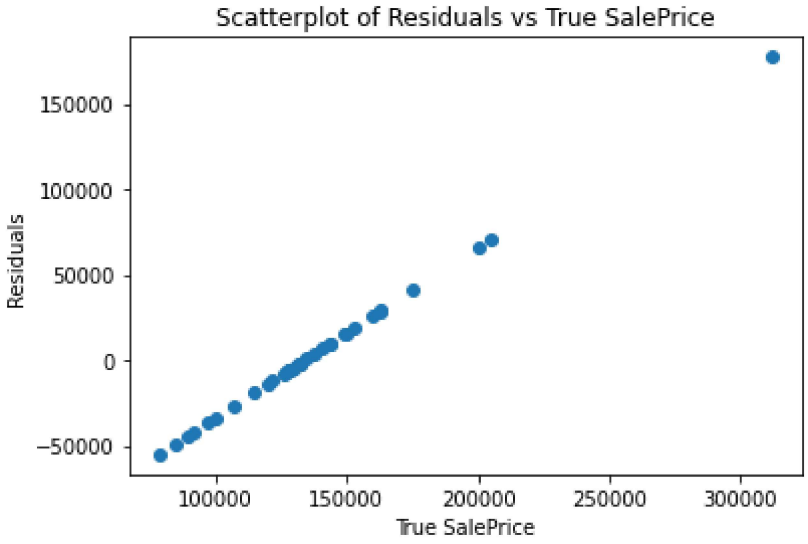
4.3

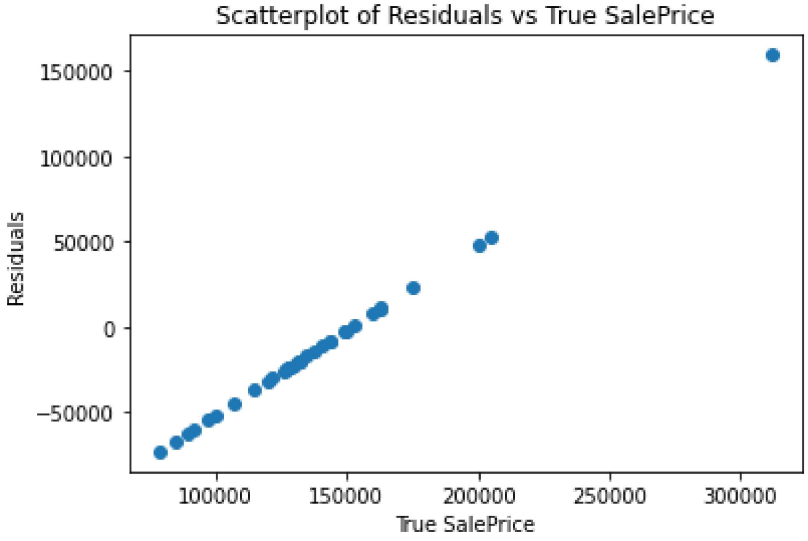
How reliable are your predictions?

In [50]:

```
#Plot a scatterplot of the predicted vs the true value of the SalePrice

lasso_best=Lasso(alpha=lasso_cv_model.alpha_)
lasso_best.fit(X_train, y_train)

y_val_pred=lasso_best.predict(X_val)

residuals_lasso=y_val-y_val_pred

tree_reg=DecisionTreeRegressor(max_depth=5, random_state=0)
tree_reg=tree_reg.fit(X_train,y_train)

y_val_pred=tree_reg.predict(X_val)

residuals_tree=y_val-y_val_pred

knn = KNeighborsRegressor(n_neighbors=8)
knn.fit(X_train,y_train)

y_val_pred=knn.predict(X_val)

residuals_knn=y_val-y_val_pred

plt.scatter(y_val, residuals_lasso)
plt.xlabel("True SalePrice")
plt.ylabel("Residuals")
plt.title("Scatterplot of Residuals vs True SalePrice")
plt.show()

plt.scatter(y_val, residuals_tree)
plt.xlabel("True SalePrice")
plt.ylabel("Residuals")
plt.title("Scatterplot of Residuals vs True SalePrice")
plt.show()

plt.scatter(y_val, residuals_knn)
plt.xlabel("True SalePrice")
plt.ylabel("Residuals")
plt.title("Scatterplot of Residuals vs True SalePrice")
plt.show()
```



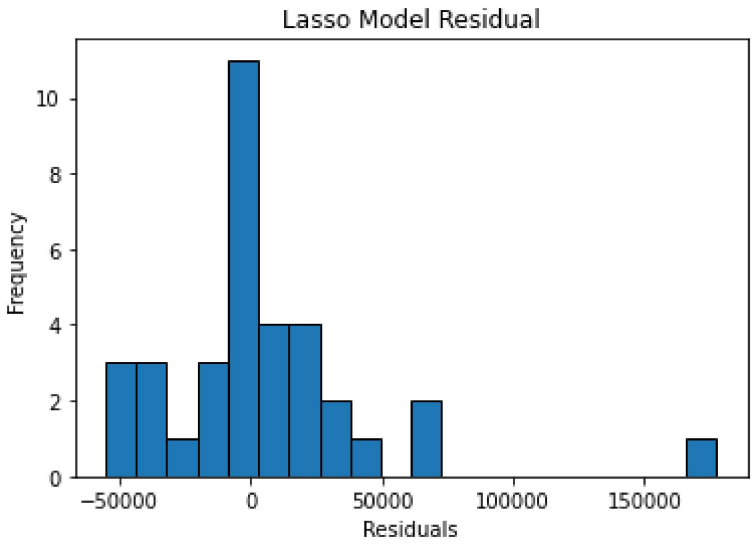


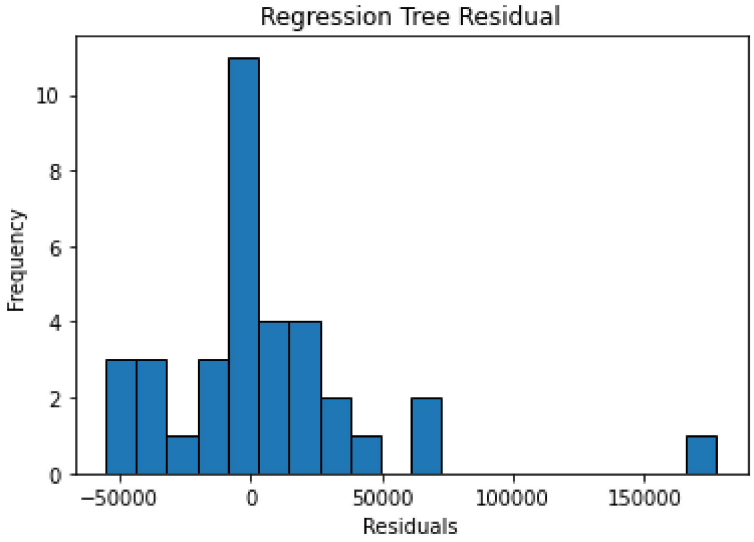


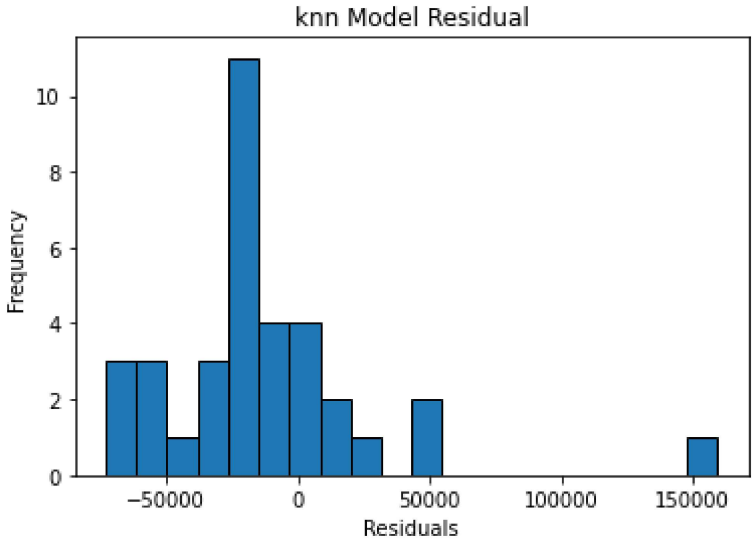
A histogram of errors could also give a good insight on any underlying patterns

In [51]:

```
#Plot a histogram of the residuals.

plt.hist(residuals_lasso, bins=20, edgecolor='black')
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Lasso Model Residual")
plt.show()

plt.hist(residuals_tree, bins=20, edgecolor='black')
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Regression Tree Residual")
plt.show()

plt.hist(residuals_knn, bins=20, edgecolor='black')
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("knn Model Residual")
plt.show()
```






knn Model Residual
Frequency
Residuals
0
2
4
6
8
10
−50000
0
50000
100000
150000