BUAN 6341.0U1 - Group 20

Machine Learning Project (REGRESSION)

Group Members:

PAS180003 - Priyanka Arvind Savant

TXD180011 - Taniya Dhar

Data Set Information:

This dataset is taken from kaggle.

This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015. The goal is to predict the house sale price in King County by building regression models that can fit the dataset the best. The dataset consists of 21613 rows and 21 columns. Consider price as the target column for regression tasks.

Importing key Libraries.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

To make sure that the whole output is printed and to handle warnings.

In [3]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
house_predict = pd.read_csv("house_predict.csv")
```

In [5]:
```
house_predict.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
id               21613 non-null int64
date             21613 non-null object
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21613 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21613 non-null float64
waterfront       21613 non-null int64
view             21613 non-null int64
condition        21613 non-null int64
grade            21613 non-null int64
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
lat              21613 non-null float64
long             21613 non-null float64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

In [6]:
```
#checking for null values
house_predict.isnull().sum()
```

Out[6]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [7]:
```
house_predict.shape
```

Out[7]:
```
(21613, 21)
```

Exploratory Data Analysis

First , let us analyze house_predict.

In [8]:
```
house_predict.dtypes
```

Out[8]:
```
id                 int64
date              object
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

1.house_predict contains 21613 rows and 21 columns. 2.All columns contain 21613 non-null values. 3.Datatype breakdown: a.Columns 'price','bathrooms','floors','lat',and 'long' are float64. b.Column 'date' is of object datatype and the remaining are int64.

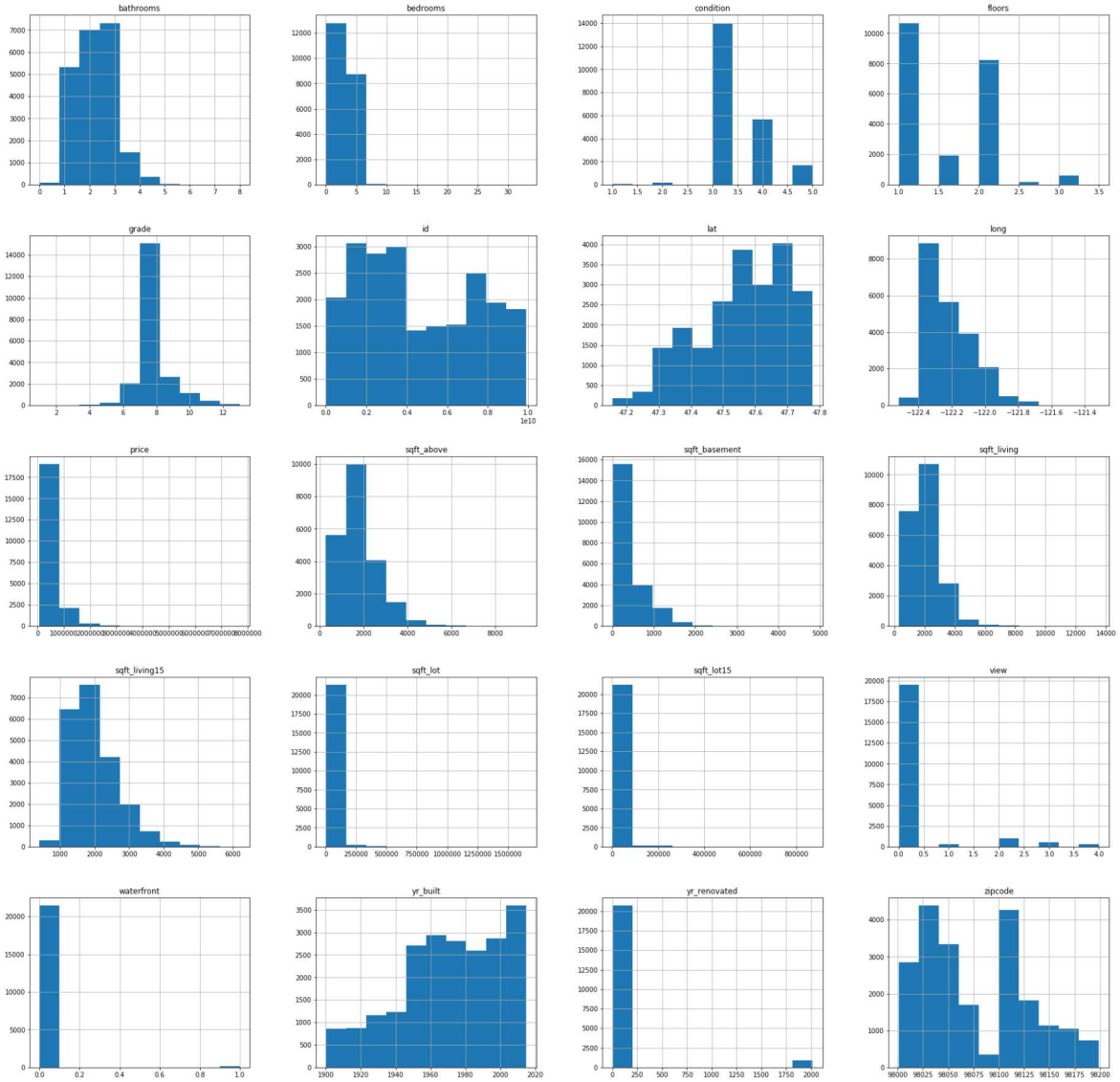
In [9]:

```
# we checked for null values in the columns they aren't any, now,we check for
#1. The date column can be removed as it as no contribution to the prices.
#2. for remaining columns we need to visualize them first induvidually.
#3. Need to be clear about data dictonary and careful while removing columns
print(house_predict['waterfront'].value_counts())
print(house_predict['view'].value_counts())
print(house_predict['condition'].value_counts())
```

```
0    21450
1      163
Name: waterfront, dtype: int64
0    19489
2      963
3      510
1      332
4      319
Name: view, dtype: int64
3    14031
4     5679
5     1701
2      172
1       30
Name: condition, dtype: int64
```

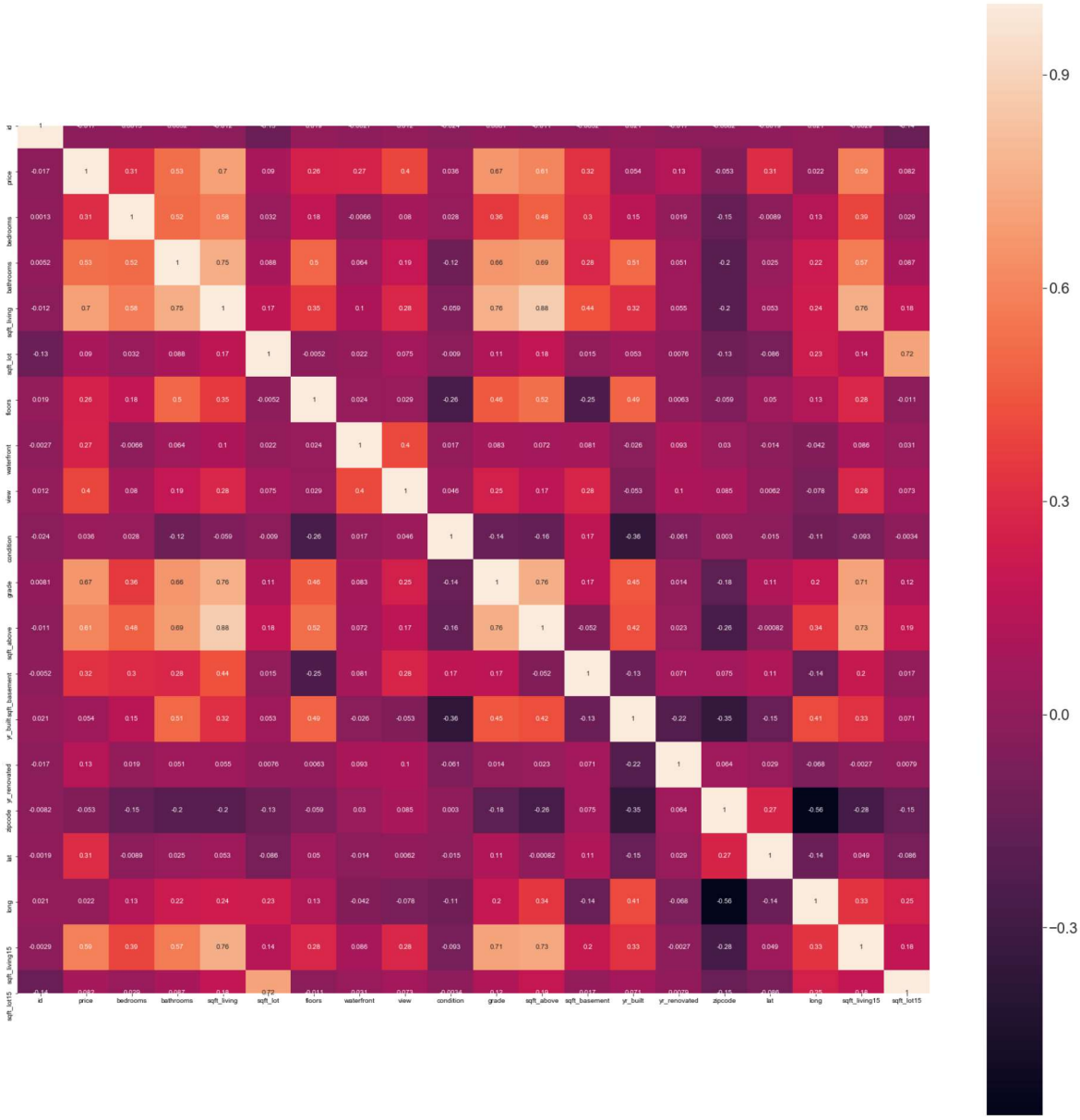
In [10]:
```
house_predict.hist(sharex=False, sharey=False, figsize=(30, 30), xlabelsize=1
plt.show()
```

Out[10]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000023F681374A8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F684F3F28
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F685304E0
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68B22A90
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68B60080
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68B90630
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68BC0BE0
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68BFF1D0
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68BFF208
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68C5FCF8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68C9E2E8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68CCB898
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68CFFE48
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68D3D400
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68D6C9B0
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68DA1F60
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68DDB550
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68E0DB00
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68E4A0F0
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023F68E7B6A0
>]],
      dtype=object)
```


bathrooms
bedrooms
condition
floors
grade
id
1e10
lat
long
price
sqft_above
sqft_basement
sqft_living
sqft_living15
sqft_lot
sqft_lot15
view
waterfront
yr_built
yr_renovated
zipcode

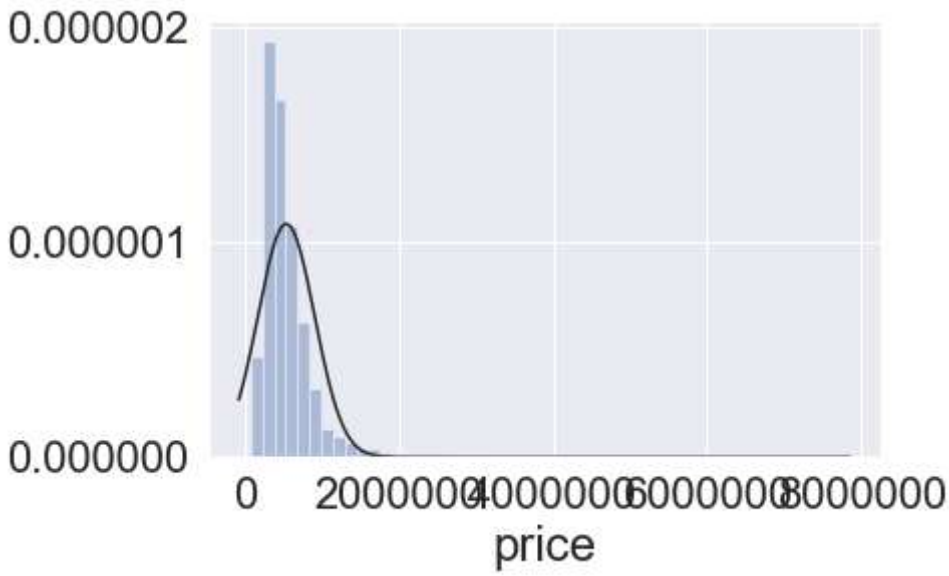
In [11]:
```
#correlation_heatmap
import seaborn as sns
fig = plt.subplots(figsize = (30,30))
sns.set(font_scale=2.0)
sns.heatmap(house_predict.corr(),square = True,cbar=True,annot=True,annot_kws
plt.show()
```

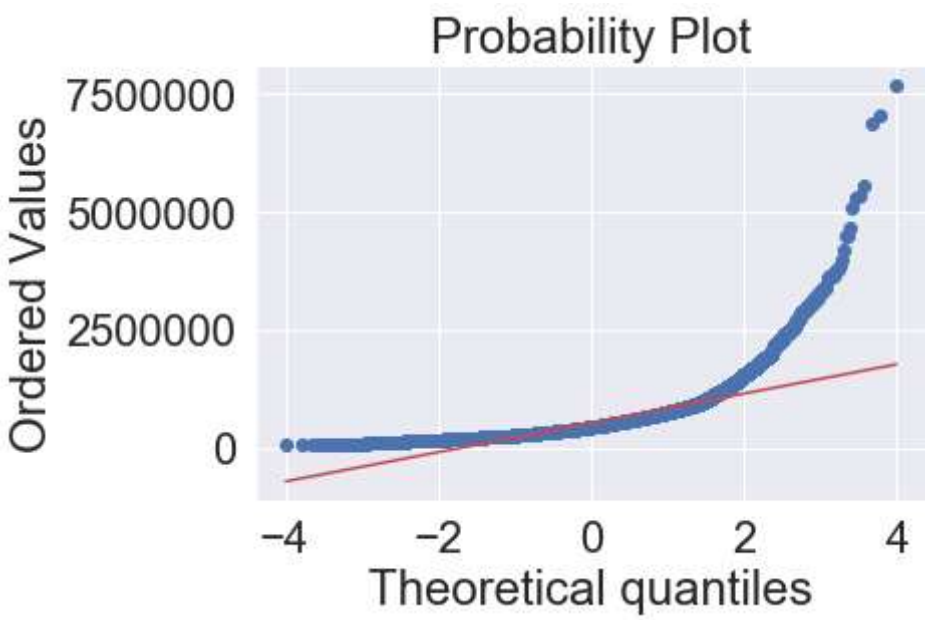
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x23f6afd2710>



In [12]:
```
from scipy.stats import norm
from scipy import stats
import seaborn as sns
#histogram and normal probability plot
sns.distplot(house_predict['price'], fit=norm, bins=50, kde=False);
fig = plt.figure()
res = stats.probplot(house_predict['price'], plot=plt)
```

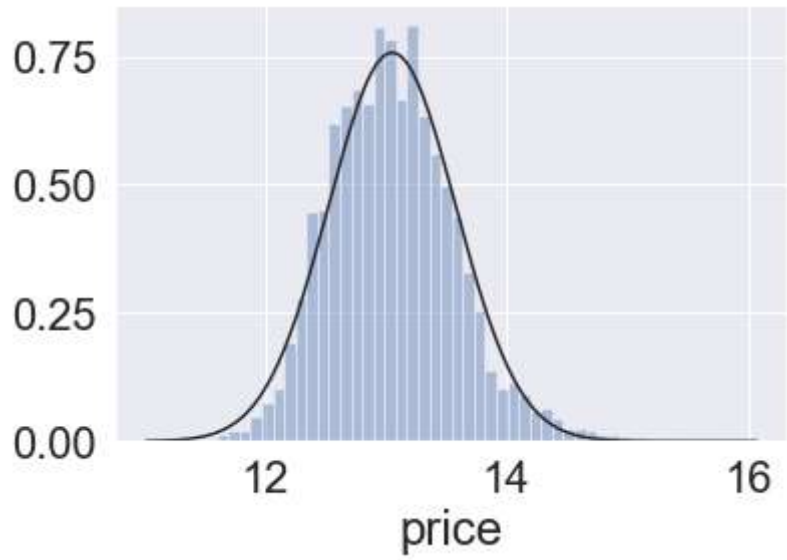
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x23f6b11a3c8>

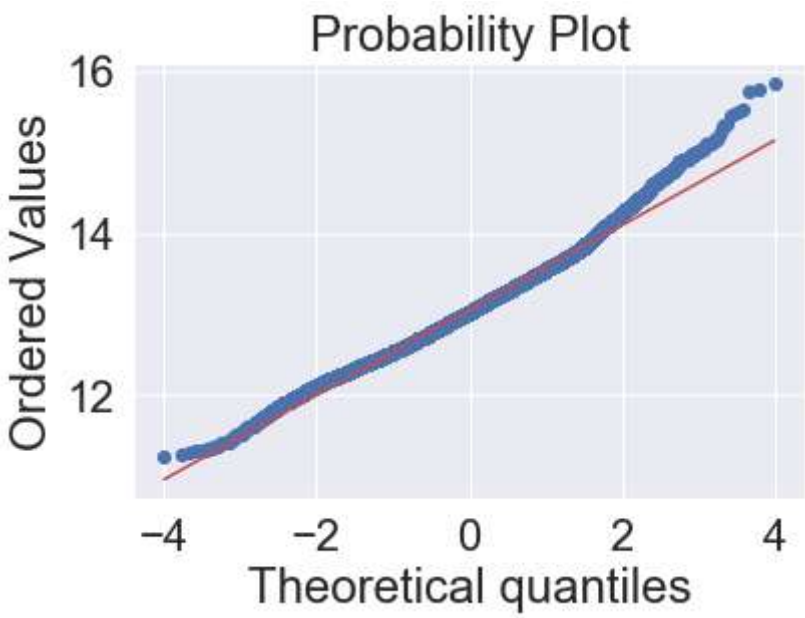




In [13]:
```
house_predict['price'] = np.log(house_predict['price'])
sns.distplot(house_predict['price'], fit=norm,  bins=50, kde=False);
fig = plt.figure()
res = stats.probplot(house_predict['price'], plot=plt)
```

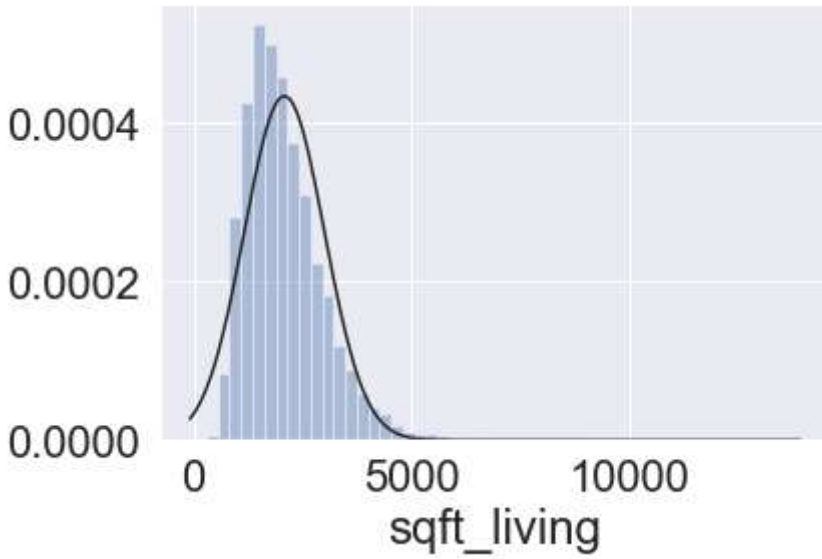
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x23f6c6aabe0>

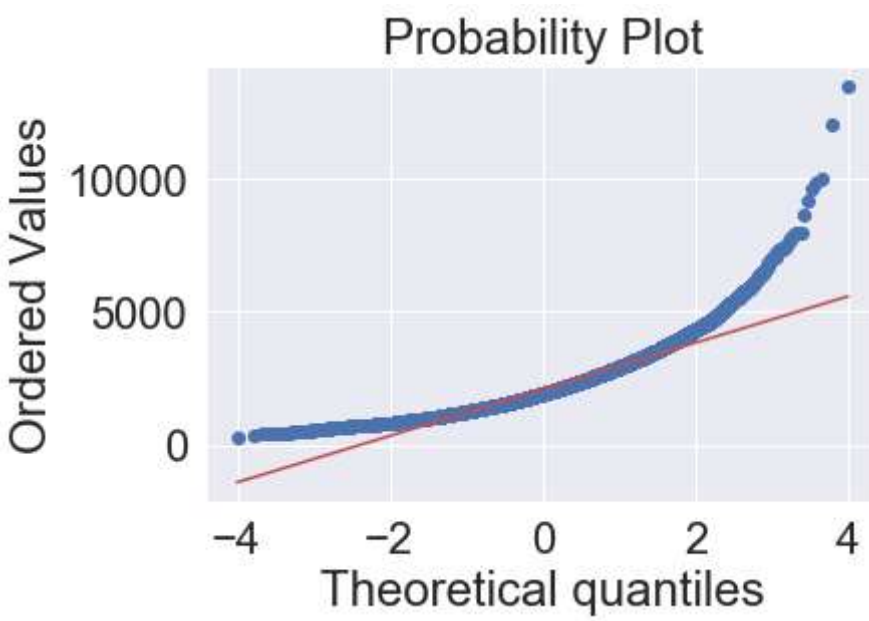




In [14]:
```
#histogram and normal probability plot
sns.distplot(house_predict['sqft_living'], fit=norm, bins=50, kde=False);
fig = plt.figure()
res = stats.probplot(house_predict['sqft_living'], plot=plt)
```

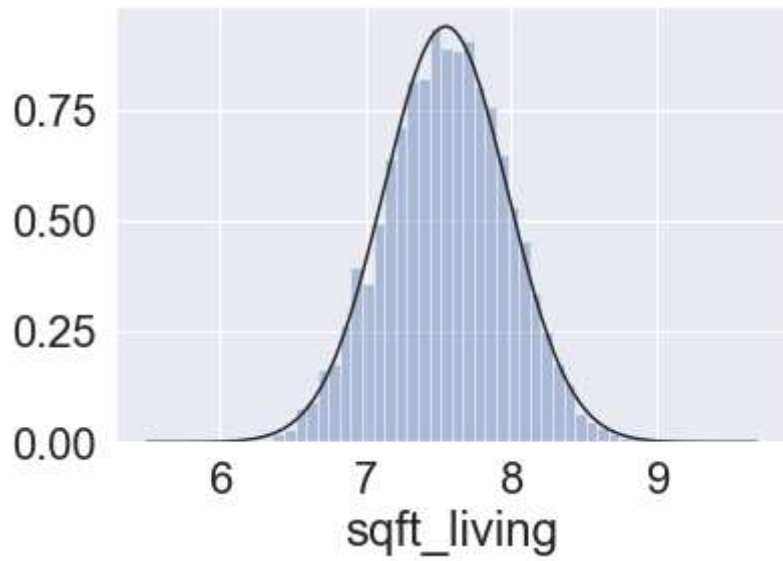
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x23f6c80bcf8>

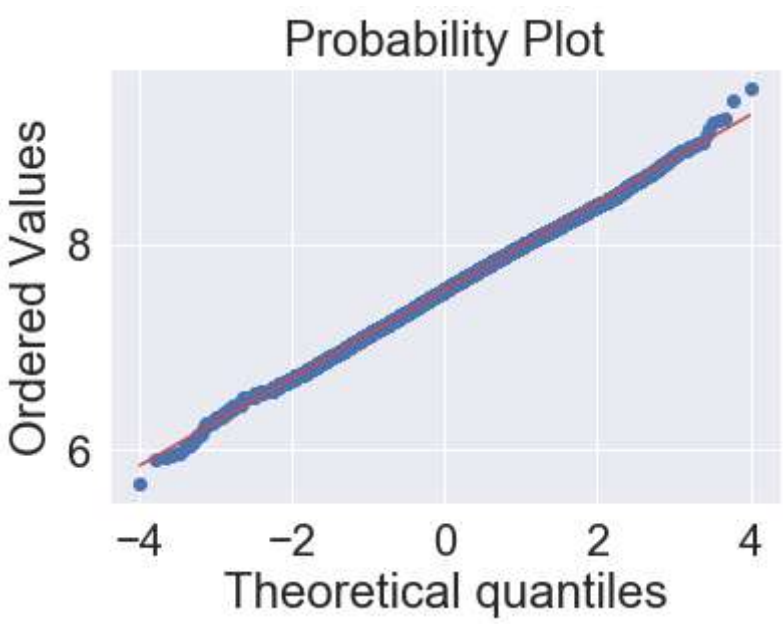




In [15]:
```
house_predict['sqft_living'] = np.log(house_predict['sqft_living'])
sns.distplot(house_predict['sqft_living'], fit=norm,  bins=50, kde=False);
fig = plt.figure()
res = stats.probplot(house_predict['sqft_living'], plot=plt)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x23f6c9382e8>





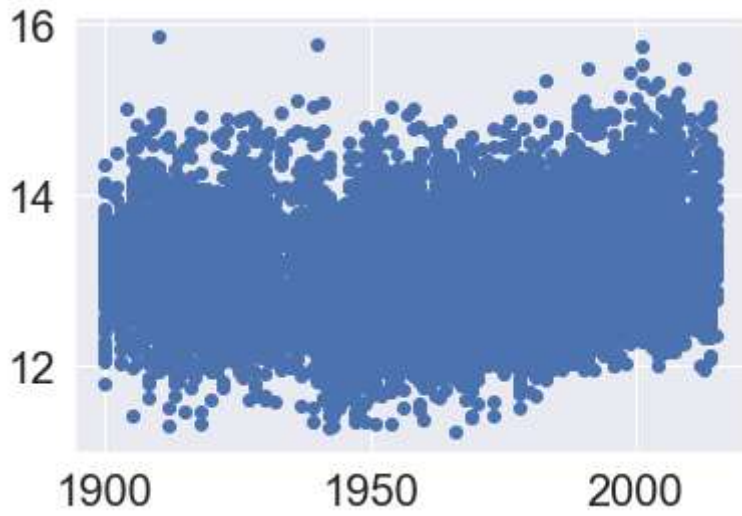
In [16]:
```
#'waterfront','view','lat','long'
predict=house_predict.drop(['id','date','zipcode'], axis=1)
```

In [17]:
```
if(predict['sqft_living'].all()==predict['sqft_above'].all()+predict['sqft_ba
    print('1')
```

```
1
```

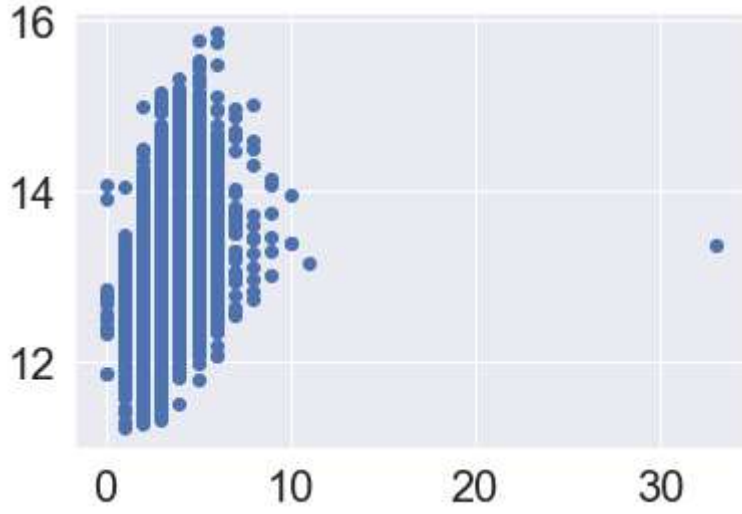
In [18]:
```
plt.scatter(predict['yr_built'],predict['price'])
plt.show()
```

Out[18]:
```
<matplotlib.collections.PathCollection at 0x23f687a8160>
```



In [19]:
```
plt.scatter(predict['bedrooms'],predict['price'])
plt.show()
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x23f6cbdbb70>
```



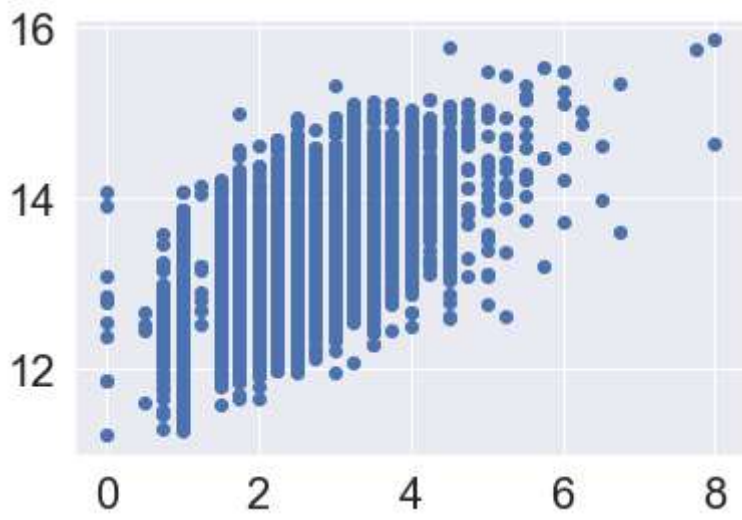
In [ ]:
```
predict = predict[predict.bedrooms < 9]
```

In [ ]:
```
predict['bedrooms'].value_counts()
```

Out[21]:
```
3    9824
4    6882
2    2760
5    1601
6     272
1     199
7      38
8      13
0      13
Name: bedrooms, dtype: int64
```

In [ ]:
```
plt.scatter(predict['bathrooms'],predict['price'])
```

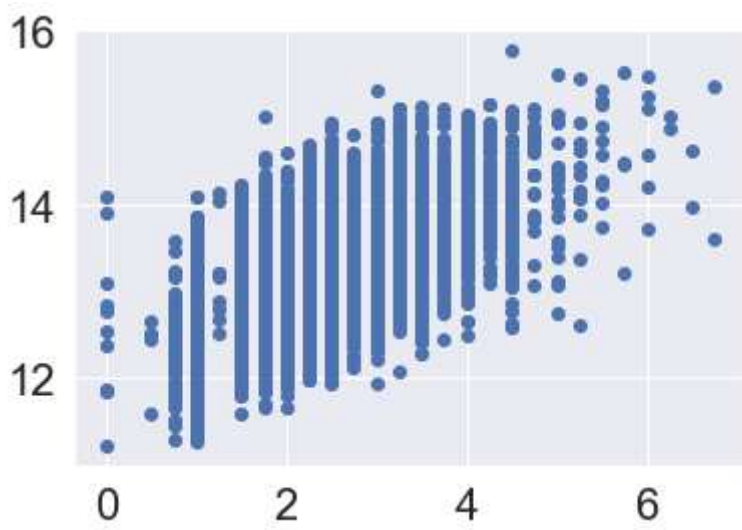
Out[22]:
```
<matplotlib.collections.PathCollection at 0x23f68594160>
```



In [ ]:
```
predict = predict[predict.bathrooms < 7 ]
```

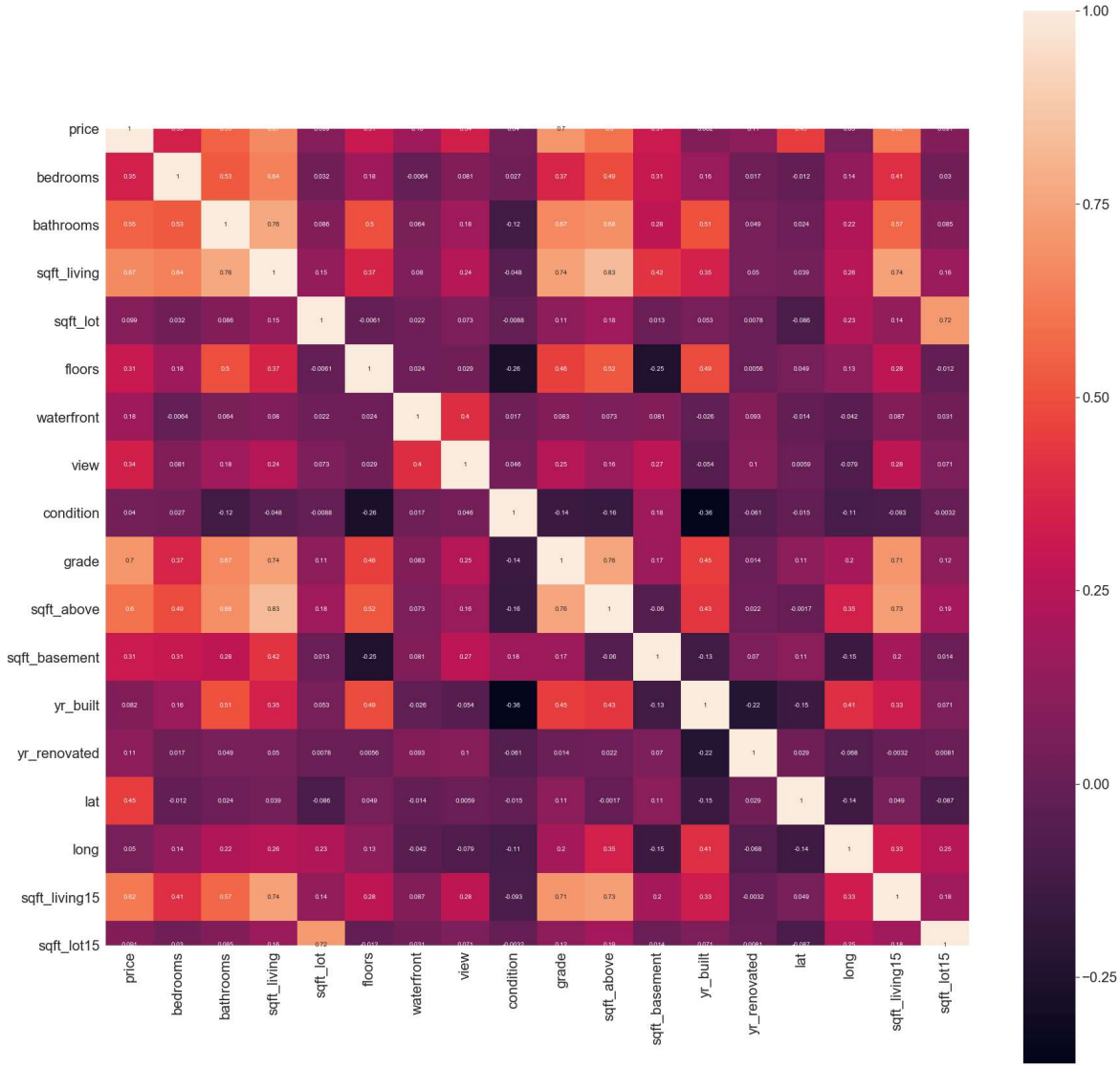
In [ ]:
```
plt.scatter(predict['bathrooms'],predict['price'])
```

Out[24]:
```
<matplotlib.collections.PathCollection at 0x23f685e6438>
```



In [ ]:
```
#correlation_heatmap
import seaborn as sns
fig = plt.subplots(figsize = (30,30))
sns.set(font_scale=2.0)
sns.heatmap(predict.corr(),square = True,cbar=True,annot=True,annot_kws={'siz
plt.show()
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x23f68594588>



In [ ]:
```
#outliers
from scipy import stats
import numpy as np
z = np.abs(stats.zscore(predict))
print(z)
predict_out = predict[(z < 4).all(axis=1)]
print(predict.shape)
print(predict_out.shape)
```

```
[[1.40169974 0.40791561 1.45384743 ... 0.30628291 0.94354818 0.26072522]
 [0.28194848 0.40791561 0.17879808 ... 0.74649619 0.43250404 0.18780241]
 [1.7995367  1.52176759 1.45384743 ... 0.13587777 1.07142587 0.17229395]
 ...
 [0.27155815 1.52176759 1.78037653 ... 0.6044919  1.41078854 0.3942887 ]
 [0.28151739 0.40791561 0.50532718 ... 1.02855737 0.84133936 0.42068609]
 [0.67625443 1.52176759 1.78037653 ... 0.6044919  1.41078854 0.41811968]]
(21599, 18)
(19868, 18)
```

In [ ]:
```
#X = pd.DataFrame(data.data, columns=data.feature_na)
#Target name
y = predict_out['price']
```

In [ ]:
```
y = np.log(predict_out['price'])
X = predict_out.drop("price",axis=1)
```

In [ ]:
```
X = predict_out.drop("price", axis = 1)
X = X.drop("bathrooms", axis = 1)
```

In [ ]:
```
X = X.drop("yr_built",axis = 1)
```

21597	4	8.134468	10125	2.0	0	0	3	10	341
21598	4	8.044947	7866	2.0	0	2	3	9	311
21599	4	8.291547	7838	2.0	0	0	3	9	399
21600	5	8.405144	8088	2.0	0	0	3	11	447
21601	3	7.261927	1179	3.0	0	0	3	8	142
21602	3	7.313220	11968	1.0	0	0	3	6	150
21603	3	7.727535	5536	2.0	0	0	3	8	227
21604	3	7.306531	1126	3.0	0	0	3	8	149
21605	4	7.832014	6023	2.0	0	0	3	9	252
21606	4	8.163371	7200	2.0	0	0	3	9	260
21607	3	7.177782	1294	2.0	0	0	3	8	118

In [ ]:
```
X.shape
```

Out[31]: (19868, 15)

In [ ]:
```
from sklearn.model_selection import train_test_split
#step 1: split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

In [ ]:
```
X_train.shape
```

Out[33]:
```
(14901, 15)
```

Scaling and Justification for type of scaling used

Next, we can prepare the data for regression by scaling 'X'. The scaling method we are using is the Standard Scaler since the outliers have already been removed. Since a majority of the values in each column are very small,the data would unnecessarily be skewed if we use min-max scaling and so, we decide to use Standard Scaling.

In [ ]:
```
# preprocessing using zero mean and unit variance scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
C:\Users\arams\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:62
5: DataConversionWarning: Data with input dtype int64, float64 were all con
verted to float64 by StandardScaler.
  return self.partial_fit(X, y)
```

Out[34]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
C:\Users\arams\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: DataCon
versionWarning: Data with input dtype int64, float64 were all converted to
float64 by StandardScaler.
  """
C:\Users\arams\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: DataCon
versionWarning: Data with input dtype int64, float64 were all converted to
float64 by StandardScaler.
```

Regression Task:

1.Apply any two models with bagging and any two models with pasting. 2.Apply any two models with adaboost boosting 3.Apply one model with gradient boosting 4.Apply PCA on data and then apply all the models in project 1 again on data you get from PCA. Compare your results with results in project 2. You don't need to apply all the models twice. Just copy the result table from project 1, prepare similar table for all the models after PCA and compare both tables. Does PCA help in getting better results? 5.Apply deep learning models covered in class

Task 1: Apply any two models with bagging and any two models with pasting.

Bagging Model 1: Lasso

In [ ]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [100,200,250,500],'max_samples':[50,75,100,150,

grid_search = GridSearchCV(BaggingRegressor(Lasso(alpha=0.01),bootstrap=True,
grid_search.fit(X_train,y_train)
```

In [ ]:
```
print("parameters",grid_search.best_params_)
```

In [ ]:
```
bag_reg = BaggingRegressor(Lasso(alpha=0.01), n_estimators=500, max_samples=2
bag_reg.fit(X_train,y_train)
btr_score = bag_reg.score(X_train,y_train)
bte_score = bag_reg.score(X_test,y_test)
oob = bag_reg.oob_score_
```

In [ ]:
```
print(btr_score)
print(bte_score)
print(oob)
```

Bag2:Ridge

In [ ]:
```
from sklearn.ensemble import BaggingRegressor
from  sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [100,200,250,500],'max_samples':[50,75,100,150,

grid_search = GridSearchCV(BaggingRegressor(Ridge(alpha=10),bootstrap=True,oo
grid_search.fit(X_train,y_train)
```

In [ ]:
```
print("parameters",grid_search.best_params_)
```

In [ ]:
```
from  sklearn.linear_model import Ridge
bag_reg = BaggingRegressor(Ridge(alpha=10), n_estimators=500, max_samples=200
bag_reg.fit(X_train,y_train)
btr_score = bag_reg.score(X_train,y_train)
bte_score = bag_reg.score(X_test,y_test)
oob = bag_reg.oob_score_
```

In [ ]:
```
print(btr_score)
print(bte_score)
print(oob)
```

Paste 1 : DecisionTree

In [ ]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [100,200,250,500],'max_samples':[50,75,100,150,

grid_search = GridSearchCV(BaggingRegressor(DecisionTreeRegressor(splitter="r
grid_search.fit(X_train,y_train)
```

In [ ]:
```
print("parameters",grid_search.best_params_)
```

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor
bag_reg = BaggingRegressor(DecisionTreeRegressor(splitter="random", max_leaf_
bag_reg.fit(X_train,y_train)
btr_score = bag_reg.score(X_train,y_train)
bte_score = bag_reg.score(X_test,y_test)
```

In [ ]:
```
print(btr_score)
print(bte_score)
```

Paste 2 : Linear Regression

In [ ]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [100,200,250,500],'max_samples':[50,75,100,150,

grid_search = GridSearchCV(BaggingRegressor(LinearRegression(),bootstrap=Fals
grid_search.fit(X_train,y_train)
```

In [ ]:
```
print("parameters",grid_search.best_params_)
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
bag_reg = BaggingRegressor(LinearRegression(), n_estimators=500, max_samples=
bag_reg.fit(X_train,y_train)
btr_score = bag_reg.score(X_train,y_train)
bte_score = bag_reg.score(X_test,y_test)
```

In [ ]:
```
print(btr_score)
print(bte_score)
```

Task 2: Apply any two models with adaboost boosting.

Ada Boost Model 1: Linear SVR

In [ ]:
```
from sklearn.svm import LinearSVR
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import cross_val_score , GridSearchCV

n_estimators_val = [10, 50, 100]
learning_rate_val = [0.1, 0.5, 1]

param_grid = dict(n_estimators = n_estimators_val, learning_rate=learning_rat

ada_reg = AdaBoostRegressor(LinearSVR(C = 0.5, max_iter=5000), loss='linear',

adaboost_regressor = GridSearchCV(estimator=ada_reg, param_grid = param_grid
adaboost_regressor.fit(X_train, y_train)
print ('Best Parameters: ',adaboost_regressor.best_params_)
```

In [ ]:
```
from sklearn.ensemble import AdaBoostRegressor

ada_reg = AdaBoostRegressor(LinearSVR(C = 0.5, max_iter=50000), n_estimators=
ada_reg.fit(X_train, y_train)
adtr_score = ada_reg.score(X_train,y_train)
adte_score = ada_reg.score(X_test,y_test)
```

In [ ]:
```
print(adtr_score)
print(adte_score)
```

Ada Boost Model 2 : KNN Regressor

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import cross_val_score , GridSearchCV

n_estimators_val = [10, 50, 100]
learning_rate_val = [0.1, 0.5, 1]

param_grid = dict(n_estimators = n_estimators_val, learning_rate=learning_rat

ada_reg = AdaBoostRegressor(KNeighborsRegressor(n_neighbors=55),loss='exponer

adaboost_regressor = GridSearchCV(estimator=ada_reg, param_grid = param_grid
adaboost_regressor.fit(X_train, y_train)
print ('Best Parameters: ',adaboost_regressor.best_params_)
```

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
ada_reg = AdaBoostRegressor(KNeighborsRegressor(n_neighbors=55), n_estimators
ada_reg.fit(X_train, y_train)
adtr_score = ada_reg.score(X_train,y_train)
adte_score = ada_reg.score(X_test,y_test)
```

In [ ]:
```
print(adtr_score)
print(adte_score)
```

3.Apply one model with gradient boosting

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor

n_estimators_val = [50, 100, 200, 500]
learning_rate_val = [0.01, 0.1, 0.5, 1]
max_depth_val = [1, 2, 3, 4]

param_grid = dict(n_estimators = n_estimators_val, learning_rate=learning_rat
gbrg = GradientBoostingRegressor(random_state=0)

gradientboost_regressor = GridSearchCV(estimator=gbrg, param_grid=param_grid)
gradientboost_regressor.fit(X_train, y_train)

print (gradientboost_regressor.best_params_)
print('Train score: {:.2f}'.format(gradientboost_regressor.score(X_train, y_t
print('Test score: {:.2f}'.format(gradientboost_regressor.score(X_test, y_tes
```

4. Apply PCA on data and then apply all the models in project 1 again on data you get from PCA.

1st we run the models in Project-1 again (without PCA).

Linear Regression

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
kfold =KFold(n_splits=3, random_state=0)

lreg = LinearRegression()
lreg.fit(X_train, y_train)
mn = 'Linear Regression'
cv_train=cross_val_score(lreg, X_train, y_train, cv=kfold).mean()
test=lreg.score(X_test, y_test)
print('Train score: %.4f'%cross_val_score(lreg, X_train, y_train, cv=kfold).m
print('Test score: %.4f'%lreg.score(X_test, y_test))
```

Now we append the scores to a dataframe called 'mr' which stands for 'Model_Results' for readablity.

In [ ]:
```
pre_pca = pd.DataFrame(columns=('S.No','Model_Name','Parameters', 'Train_Scor
```

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[1,mn,'none',cv_train,test]
```

In [ ]:
```
pre_pca
```

Lasso Regression

In [ ]:
```
from sklearn.linear_model import Lasso
kfold =KFold(n_splits=3, random_state=0)
param_grid={'alpha':[0.0001,0.001,0.01, 0.1, 0.5, 1, 5, 10]}
grid_search = GridSearchCV(Lasso(), param_grid, cv=kfold, return_train_score=
grid_search.fit(X_train,y_train)
grid_search_test=grid_search.score(X_test,y_test)
```

In [ ]:
```
print("parameters",grid_search.best_params_)
print("Train:",grid_search.best_score_)
print("Test",grid_search_test)
```

Appending the scores to 'Model Results' for readablity.

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[2,'Lasso Regression',grid_search.best_params_,gric
```

In [ ]:
```
pre_pca
```

We can now understand by plotting the relation between the predicted values and observed values.

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

KNN Regressor

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
kfold =KFold(n_splits=5, random_state=0)
param_grid={'n_neighbors':range(1,10)}

grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, cv=kfold, retur
grid_search.fit(X_train,y_train)

knn_ts=grid_search.score(X_test, y_test)
```

In [ ]:
```
print("Train:",grid_search.best_score_)
print("Test",knn_ts)
print("parameters",grid_search.best_params_)
```

As we can see, The best K value for K nearest neighbors is 9.

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[3,'KNN Regression',grid_search.best_params_,grid_s
```

In [ ]:
```
pre_pca
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

Ridge Regression

In [ ]:
```
from  sklearn.linear_model import Ridge
kfold =KFold(n_splits=3, random_state=0)
param_grid={'alpha':[0.001,0.01, 0.1, 1, 10, 100]}

grid_search = GridSearchCV(Ridge(), param_grid, cv=kfold, return_train_score=
grid_search.fit(X_train,y_train)
test_ri=grid_search.score(X_test,y_test)
```

In [ ]:
```
print("Train:",grid_search.best_score_)
print("Test",test_ri)
print("parameters",grid_search.best_params_)
```

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[4,'Ridge Regression',grid_search.best_params_,gric
```

In [ ]:
```
pre_pca
```

In [ ]:
```
%matplotlib inline
import numpy as np

x_range1 = np.linspace(0.001, 1, 100).reshape(-1,1)
x_range2 = np.linspace(1, 10000, 10000).reshape(-1,1)

x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    coeff.append(ridge.coef_ )

coeff = np.array(coeff)

for i in range(0,13):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

Polynomial Regression

In [ ]:
```
from  sklearn.preprocessing  import PolynomialFeatures
kfold =KFold(n_splits=5, random_state=0)
train_s_list = []
test_s_list = []
for n in range(1,5):
    poly = PolynomialFeatures(n)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)
    train_s_list.append(cross_val_score(lreg, X_train_poly, y_train,cv=kfold)
    lreg.fit(X_train_poly, y_train)
    test_s_list.append(lreg.score(X_test_poly, y_test))
```

In [ ]:
```
print(train_s_list)
print(test_s_list)
```

The best scores are rendered by polynomials of degree 3.

In [ ]:
```
print("Train",train_s_list[1])
print("Test",test_s_list[1])
print("Best Parameters n=",3)
```

Adding the scores to the model results

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[5,'Polynomial Regression',2,train_s_list[1],test_s
```

In [ ]:
```
pre_pca
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

Linear SVR Model

In [ ]:
```
from sklearn.svm import LinearSVR
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=5, random_state=0)

param_grid = {'C': [0.001, 0.01, 0.1, 1, 0.5, 10, 100], 'max_iter':[50000]}
grid_search = GridSearchCV(LinearSVR(), param_grid, cv=kfold, return_train_sc

grid_search.fit(X_train, y_train)

test=grid_search.score(X_test, y_test)
```

In [ ]:
```
print("Train set score with best parameters: {:.5f}".format(grid_search.best_
print("Test set score with best parameters: {:.5f}".format(test))
print("best parameters are:",grid_search.best_params_)
```

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[6,'Linear SVR Model',grid_search.best_params_,gric
```

In [ ]:
```
pre_pca
```

SVR with Kernel Trick

In [ ]:
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=3, random_state=0)
param_grid = {'C': [0.001, 0.01, 0.1, 1, 250, 10, 100], 'max_iter':[60000]}
grid_search = GridSearchCV(SVR(kernel='linear'), param_grid, cv=kfold, return

grid_search.fit(X_train, y_train)

test=cross_val_score(grid_search,X_test, y_test,scoring='r2',cv=kfold).mean()
train=grid_search.score(X_train, y_train)
```

In [ ]:
```
print("Train set score with best parameters: {:.5f}".format(grid_search.best_
print("Test set score with best parameters: {:.5f}".format(test))
print("best parameters are:",grid_search.best_params_)
```

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[7,'SVR- Kernel (Linear)',grid_search.best_params_,
```

In [ ]:
```
pre_pca
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

SVR - Polynomial Kernel Trick

In [ ]:
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=10, random_state=0)
param_grid = {'C': [0.01, 10, 0.1, 1, 250, 100],'degree':[1,2,3,4]}
grid_search = GridSearchCV(SVR(kernel='poly'), param_grid, cv=kfold, return_t

grid_search.fit(X_train, y_train)

train=cross_val_score(grid_search,X_train, y_train,scoring='r2',cv=kfold).mea
test=grid_search.score(X_test, y_test);
```

In [ ]:
```
print("Train set score with best parameters: {:.5f}".format(train))
print("Test set score with best parameters: {:.5f}".format(test))
print("best parameters are:",grid_search.best_params_)
```

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[8,'Linear SVR- Polynomial Kernel',grid_search.best
```

In [ ]:
```
pre_pca
```

SVR RBF Kernel

In [ ]:
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=5, random_state=0)
param_grid = {'C': [0.001, 0.01, 0.1,1, 10, 100, 250,500],'gamma': [0.001, 0.
grid_search = GridSearchCV(SVR(kernel='rbf'), param_grid, cv=kfold, return_tr

grid_search.fit(X_train, y_train)

train=cross_val_score(grid_search,X_train, y_train,scoring='r2',cv=kfold).mea
test=grid_search.score(X_test, y_test);
```

In [ ]:
```
print("Train set score with best parameters: {:.5f}".format(train))
print("Test set score with best parameters: {:.5f}".format(test))
print("best parameters are:",grid_search.best_params_)
```

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[9,'SVR- RBF Kernel',grid_search.best_params_,trair
```

In [ ]:
```
pre_pca
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

SGD Regressor

In [ ]:
```
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=10, random_state=0)
param_grid = {'alpha': [0.01, 10, 0.1, 1, 250, 100, 500], 'penalty': ['l1','l
grid_search = GridSearchCV(SGDRegressor(), param_grid, cv=kfold, return_trair

grid_search.fit(X_train, y_train)

train=cross_val_score(grid_search,X_train, y_train,scoring='r2',cv=kfold).mea
test=grid_search.score(X_test, y_test)
```

In [ ]:
```
print(test)
print(train)
print(grid_search.best_params_)
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=5)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=5)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

In [ ]:
```
pre_pca.loc[len(pre_pca)]=[10,'SGD Regressor',grid_search.best_params_,train,
```

In [ ]:
```
pre_pca
```

Clearly, the best model seems to be KNN Regressor with value 9.

In [ ]:
```
pre_pca.to_csv('pre_pca_reg.csv')
```

Now we apply PCA

In [ ]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.97)
pca.fit(X_train)
pca.n_components_
```

In [ ]:
```
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

Linear Regression

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=3, random_state=0)

lreg = LinearRegression()
lreg.fit(X_train, y_train)
mn = 'Linear Regression'
cv_train=cross_val_score(lreg, X_train, y_train, cv=kfold).mean()
test=lreg.score(X_test, y_test)
print('Train score: %.4f'%cross_val_score(lreg, X_train, y_train, cv=kfold).m
print('Test score: %.4f'%lreg.score(X_test, y_test))
```

Now we append the scores to a dataframe called 'mr' which stands for 'Model_Results' for readablity.

In [ ]:
```
post_pca = pd.DataFrame(columns=('S.No','Model_Name','Parameters', 'Train_Sc
```

In [ ]:
```
post_pca.loc[len(post_pca)]=[1,mn,'none',cv_train,test]
```

In [ ]:
```
post_pca
```

Lasso Regression

In [ ]:
```
from sklearn.linear_model import Lasso
kfold =KFold(n_splits=3, random_state=0)
param_grid={'alpha':[0.0001,0.001,0.01, 0.1, 0.5, 1, 5, 10]}
grid_search = GridSearchCV(Lasso(), param_grid, cv=kfold, return_train_score=
grid_search.fit(X_train,y_train)
grid_search_test=grid_search.score(X_test,y_test)
```

In [ ]:
```
print("parameters",grid_search.best_params_)
print("Train:",grid_search.best_score_)
print("Test",grid_search_test)
```

Appending the scores to 'Model Results' for readablity.

In [ ]:
```
post_pca.loc[len(post_pca)]=[2,'Lasso Regression',grid_search.best_params_,gr
```

In [ ]:
```
post_pca
```

We can now understand by plotting the relation between the predicted values and observed values.

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

KNN Regressor

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
kfold =KFold(n_splits=5, random_state=0)
param_grid={'n_neighbors':range(1,10)}

grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, cv=kfold, retur
grid_search.fit(X_train,y_train)

knn_ts=grid_search.score(X_test, y_test)
```

In [ ]:
```
print("Train:",grid_search.best_score_)
print("Test",knn_ts)
print("parameters",grid_search.best_params_)
```

As we can see, The best K value for K nearest neighbors is 9.

In [ ]:
```
post_pca.loc[len(post_pca)]=[3,'KNN Regression',grid_search.best_params_,gric
```

In [ ]:
```
post_pca
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

Ridge Regression

In [ ]:
```
from  sklearn.linear_model import Ridge
kfold =KFold(n_splits=3, random_state=0)
param_grid={'alpha':[0.001,0.01, 0.1, 1, 10, 100]}

grid_search = GridSearchCV(Ridge(), param_grid, cv=kfold, return_train_score=
grid_search.fit(X_train,y_train)
test_ri=grid_search.score(X_test,y_test)
```

In [ ]:
```
print("Train:",grid_search.best_score_)
print("Test",test_ri)
print("parameters",grid_search.best_params_)
```

In [ ]:
```
post_pca.loc[len(post_pca)]=[4,'Ridge Regression',grid_search.best_params_,gr
```

In [ ]:
```
post_pca
```

In [ ]:
```
%matplotlib inline
import numpy as np

x_range1 = np.linspace(0.001, 1, 100).reshape(-1,1)
x_range2 = np.linspace(1, 10000, 10000).reshape(-1,1)

x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    coeff.append(ridge.coef_ )

coeff = np.array(coeff)

for i in range(0,13):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

Polynomial Regression

In [ ]:
```
from  sklearn.preprocessing  import PolynomialFeatures
kfold =KFold(n_splits=5, random_state=0)
train_s_list = []
test_s_list = []
for n in range(1,5):
    poly = PolynomialFeatures(n)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)
    train_s_list.append(cross_val_score(lreg, X_train_poly, y_train,cv=kfold)
    lreg.fit(X_train_poly, y_train)
    test_s_list.append(lreg.score(X_test_poly, y_test))
```

In [ ]:
```
print(train_s_list)
print(test_s_list)
```

The best scores are rendered by polynomials of degree 2.

In [ ]:
```
print("Train",train_s_list[1])
print("Test",test_s_list[1])
print("Best Parameters n=",2)
```

Adding the scores to the model results

In [ ]:
```
post_pca.loc[len(post_pca)]=[5,'Polynomial Regression',2,train_s_list[1],test
```

In [ ]:
```
post_pca
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

Linear SVR Model

In [ ]:
```
from sklearn.svm import LinearSVR
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
kfold =KFold(n_splits=5, random_state=0)

param_grid = {'C': [0.001, 0.01, 0.1, 1, 0.5, 10, 100], 'max_iter':[50000]}
grid_search = GridSearchCV(LinearSVR(), param_grid, cv=kfold, return_train_sc

grid_search.fit(X_train, y_train)

test=grid_search.score(X_test, y_test)
```

In [ ]:
```
print("Train set score with best parameters: {:.5f}".format(grid_search.best_
print("Test set score with best parameters: {:.5f}".format(test))
print("best parameters are:",grid_search.best_params_)
```

In [ ]:
```
post_pca.loc[len(post_pca)]=[6,'Linear SVR Model',grid_search.best_params_,gr
```

In [ ]:
```
post_pca
```

SVR with Kernel Trick

In [ ]:
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=3, random_state=0)
param_grid = {'C': [0.001, 0.01, 0.1, 1, 250, 10, 100], 'max_iter':[60000]}
grid_search = GridSearchCV(SVR(kernel='linear'), param_grid, cv=kfold, return

grid_search.fit(X_train, y_train)

test=cross_val_score(grid_search,X_test, y_test,scoring='r2',cv=kfold).mean()
train=grid_search.score(X_train, y_train)
```

In [ ]:
```
print("Train set score with best parameters: {:.5f}".format(grid_search.best_
print("Test set score with best parameters: {:.5f}".format(test))
print("best parameters are:",grid_search.best_params_)
```

In [ ]:
```
post_pca.loc[len(post_pca)]=[7,'SVR- Kernel (Linear)',grid_search.best_params
```

In [ ]:
```
post_pca
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

SVR - Polynomial Kernel Trick

In [ ]:
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=10, random_state=0)
param_grid = {'C': [0.01, 10, 0.1, 1, 250, 100],'degree':[1,2,3,4]}
grid_search = GridSearchCV(SVR(kernel='poly'), param_grid, cv=kfold, return_t

grid_search.fit(X_train, y_train)

train=cross_val_score(grid_search,X_train, y_train,scoring='r2',cv=kfold).mea
test=grid_search.score(X_test, y_test);
```

In [ ]:
```
print("Train set score with best parameters: {:.5f}".format(train))
print("Test set score with best parameters: {:.5f}".format(test))
print("best parameters are:",grid_search.best_params_)
```

In [ ]:
```
post_pca.loc[len(post_pca)]=[8,'Linear SVR- Polynomial Kernel',grid_search.be
```

In [ ]:
```
post_pca
```

SVR RBF Kernel

In [ ]:
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=5, random_state=0)
param_grid = {'C': [0.001, 0.01, 0.1,1, 10, 100, 250,500],'gamma': [0.001, 0.
grid_search = GridSearchCV(SVR(kernel='rbf'), param_grid, cv=kfold, return_tr

grid_search.fit(X_train, y_train)

train=cross_val_score(grid_search,X_train, y_train,scoring='r2',cv=kfold).mea
test=grid_search.score(X_test, y_test);
```

In [ ]:
```
print("Train set score with best parameters: {:.5f}".format(train))
print("Test set score with best parameters: {:.5f}".format(test))
print("best parameters are:",grid_search.best_params_)
```

In [ ]:
```
post_pca.loc[len(post_pca)]=[9,'SVR- RBF Kernel',grid_search.best_params_,tra
```

In [ ]:
```
post_pca
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=kfold)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=kfold)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

SGD Regressor

In [ ]:
```
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import GridSearchCV
kfold =KFold(n_splits=10, random_state=0)
param_grid = {'alpha': [0.01, 10, 0.1, 1, 250, 100, 500], 'penalty': ['l1','l
grid_search = GridSearchCV(SGDRegressor(), param_grid, cv=kfold, return_trair

grid_search.fit(X_train, y_train)

train=cross_val_score(grid_search,X_train, y_train,scoring='r2',cv=kfold).mea
test=grid_search.score(X_test, y_test)
```

In [ ]:
```
print(test)
print(train)
print(grid_search.best_params_)
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train,y_train, cv=5)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test,y_test, cv=5)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```

In [ ]:
```
post_pca.loc[len(post_pca)]=[10,'SGD Regressor',grid_search.best_params_,trai
```

In [ ]:
```
post_pca
```

In [ ]:
```
pre_pca
```

Does PCA help in getting better results?

#1.The train and test scores of KNN regression before PCA was 0.81 and 0.82 and the train and test scores of KNN regression after PCA was 0.81 and 0.83. So clearly, KNN regression with k value of 9 is the best model before PCA and after PCA.

#2.As we can see from the scores above,there is not a significant difference in regression before PCA and after PCA.

#3.There seems to be a marginal improvement in some models after PCA but there also seems to be a decrease in performance of some models post PCA. So we cannot conclusively say tha PCA helps in getting better results.

TASK 5 : Creating deep-learning models.

In [ ]:
```
X_train.shape
```

In [ ]:
```
#X = X.drop("sqft_basement", axis = 1)
#X = X.drop("sqft_above", axis = 1)
X = X.drop("long", axis = 1)
#X = X.drop("view", axis = 1)
X = X.drop("waterfront", axis = 1)
#X = X.drop("sqft_lot15", axis = 1)
```

In [ ]:
```
X_train.shape
```

In [ ]:
```
from sklearn.model_selection import train_test_split
#step 1: split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

In [ ]:
```
X_train.shape
```

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

In [ ]:
```
%matplotlib inline
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils

np.random.seed(0) # for reproducibility
```

1. Perceptron

In [ ]:
```
#step 1: build model
model1 = Sequential()
#input layer
model1.add(Dense(20, input_dim = 9, activation = 'relu'))
#hidden layers
#output layer
model1.add(Dense(1))

#step 2: make computational graph - compile
model1.compile(loss= 'mean_squared_error' , optimizer = 'adam',metrics = ['me

#step 3: train the model - fit
model1.fit(X_train, y_train, epochs = 150, batch_size = 300);
```

Perceptron model evaulation metrics:

In [ ]:
```
model1.evaluate(X_train, y_train)
```

In [ ]:
```
model1.evaluate(X_test, y_test)
```

R Squared score for perceptron

In [ ]:
```
train_pred=model1.predict(X_train)
pred=model1.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import r2_score

print("train",r2_score(y_train, train_pred))
print("test",r2_score(y_test,pred))
```

In [ ]:
```
import tensorflow as tf
from tensorflow import keras as ker
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

2. Multi layer perceptron

In [ ]:
```
#step 1: build model
model1 = Sequential()
#input layer
model1.add(Dense(30, input_dim = 9, activation = 'relu'))
#hidden layers
model1.add(Dense(20, activation = 'relu'))
model1.add(Dense(10, activation = 'relu'))
#output layer
model1.add(Dense(1))

#step 2: make computational graph - compile
model1.compile(loss= 'mean_squared_error' , optimizer = 'adam',metrics = ['me

#step 3: train the model - fit
model1.fit(X_train, y_train, epochs = 100, batch_size = 250);
```

Perceptron model evaulation metrics:

In [ ]:
```
model1.evaluate(X_train, y_train)
```

In [ ]:
```
model1.evaluate(X_test, y_test)
```

R Squared score for perceptron

In [ ]:
```
train_pred=model1.predict(X_train)
pred=model1.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import r2_score

print("train",r2_score(y_train, train_pred))
print("test",r2_score(y_test,pred))
```

Conclusion:

#1.Two models with bagging(Lasso Regression,Ridge Regression) and two models with pasting(Decision Tree Regression,Linear Regression) were applied.

#2.Two models with adaboost boosting(SVC Polynomial Kernel Trick, KNN Regression) were applied.

#3.One model with gradient boosting was applied.

#4.The following models were applied before PCA and after PCA:

#a.Linear Regression

#b.Lasso

#c.Ridge

#d.KNN Regression

#e.SVR - 'Linear' Kernel Trick

#f.SVR - 'Polynomial' Kernel Trick

#g.SVR - 'RBF' Kernel Trick

#h.Linear SVR

#i.Polynomial Regression

#j.SGD Regressor

#The results were compared and the explanation of whether PCA yields better results was given.

#5.Two Deep Learning models(Perceptron and Multi Layer Perceptron) were applied.

#6. The linear SVR for Adaboosting is taking alot of time to execute the code.