Regression

K-NN and Linear Regression

Session learning objectives: KNN

Session learning objective: Linear Regression

- Use Python to fit simple and multivariable linear regression models on training data.
- Evaluate the linear regression model on test data.
- Compare and contrast predictions obtained from K-nearest neighbors regression to those obtained using linear regression from the same data set.
- Describe how linear regression is affected by outliers and multicollinearity.

The regression problem

- Predictive problem
- Use past information to predict future observations
- Predict *numerical* values instead of *categorical* values

Examples:

- Race time in the Boston marathon
- size of a house to predict its sale price

Regression Methods

In this workshop:

- K-nearest neighbors
- Linear regression

Classification similarities to regression

Concepts from classification map over to the setting of regression

- Predict a new observation's response variable based on past observations
- Split the data into training and test sets
- Use cross-validation to evaluate different choices of model parameters

Difference

Predicting numerical variables instead of categorical variables

Explore a data set

932 real estate transactions in Sacramento, California

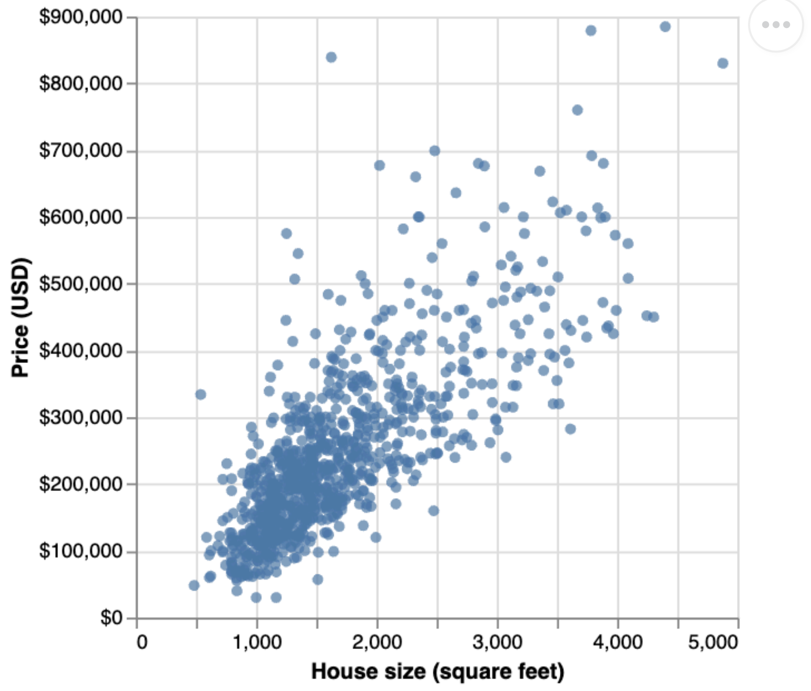
> Can we use the size of a house in the Sacramento, CA area to predict its sale price?

Data and package setup

```python
import altair as alt
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn import set_config

# Output dataframes instead of arrays
set_config(transform_output='pandas')

sacramento = pd.read_csv('data/sacramento.csv')
print(sacramento)
```

```
                city     zip  beds  baths  sqft         type   price  \
0         SACRAMENTO  z95838     2    1.0   836  Residential   59222
1         SACRAMENTO  z95823     3    1.0  1167  Residential   68212
..               ...     ...   ...    ...   ...          ...     ...
930        ELK_GROVE  z95758     4    2.0  1685  Residential  235301
931  EL_DORADO_HILLS  z95762     3    2.0  1362  Residential  235738

      latitude   longitude
0    38.631913 -121.434879
1    38.478902 -121.431028
..         ...         ...
930  38.417000 -121.397424
931  38.655245 -121.075915

[932 rows x 9 columns]
```

Price vs Sq.Ft



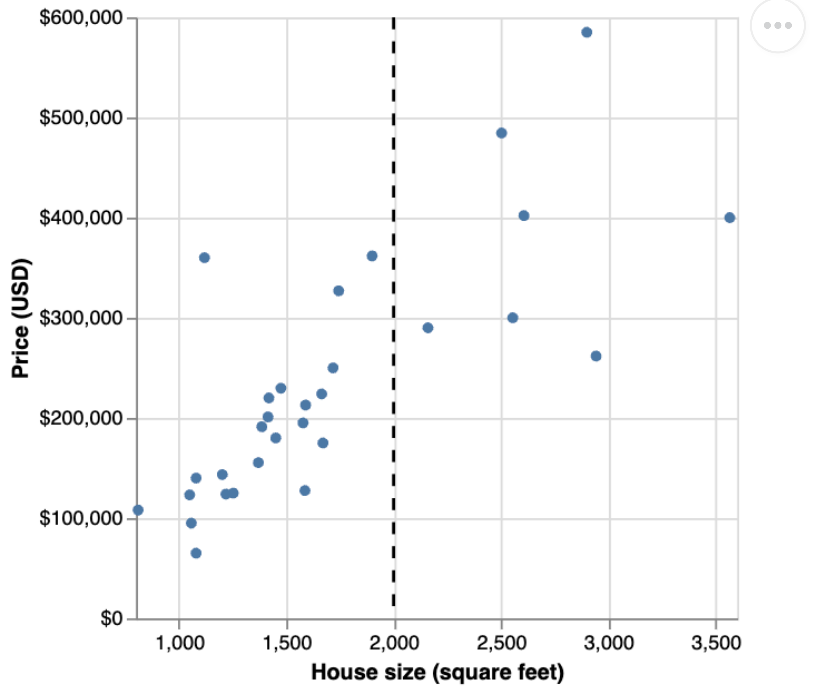
K-nearest neighbors regression

```python
# look at a small sample of data
np.random.seed(10)

small_sacramento = sacramento.sample(n=30)
print(small_sacramento)
```

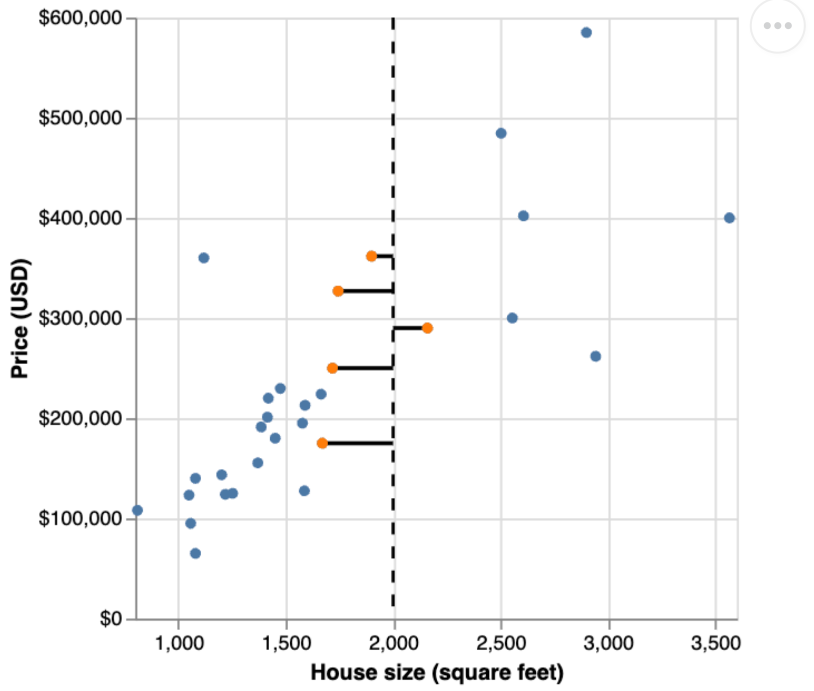
```
            city     zip  beds  baths  sqft         type   price   latitude  \
538    ELK_GROVE  z95758     3    3.0  2503  Residential  484500  38.409689
304      ROCKLIN  z95765     4    2.0  2607  Residential  402000  38.805749
..           ...     ...   ...    ...   ...          ...     ...        ...
559   SACRAMENTO  z95817     2    1.0  1080  Residential   65000  38.544162
917   SACRAMENTO  z95834     3    2.0  1665  Residential  224000  38.631026

      longitude
538 -121.446059
304 -121.280931
..          ...
559 -121.460652
917 -121.501879

[30 rows x 9 columns]
```

Sample: Example

House price of 2000



5 closest neighbors



Sample: Predict

5 closest points

```python
small_sacramento['dist'] = (2000 - small_sacramento['sqft']).abs()
nearest_neighbors = small_sacramento.nsmallest(5, 'dist')
print(nearest_neighbors)
```

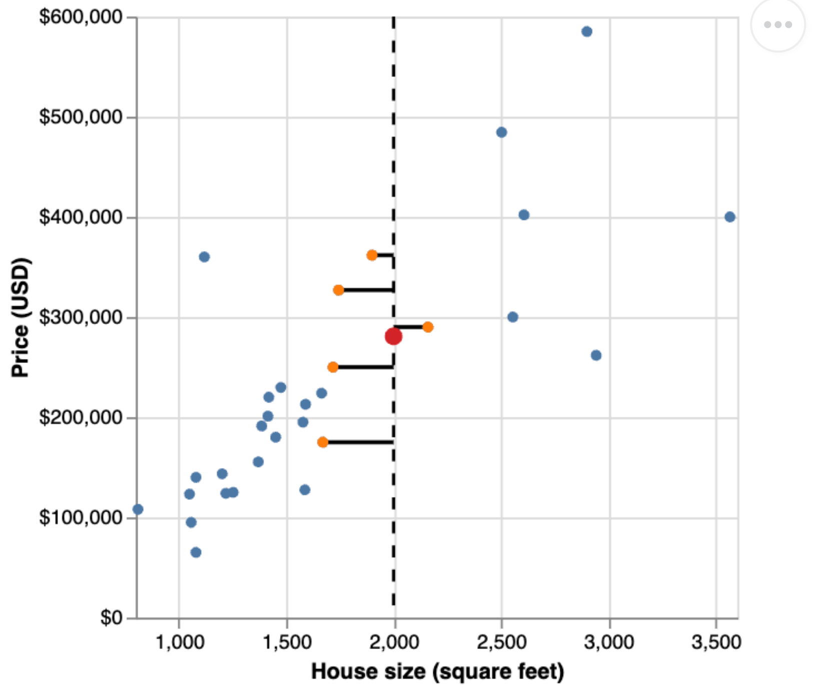
```
               city     zip  beds  baths  sqft         type   price  \
298      SACRAMENTO  z95823     4    2.0  1900  Residential  361745
718        ANTELOPE  z95843     4    2.0  2160  Residential  290000
748       ROSEVILLE  z95678     3    2.0  1744  Residential  326951
252      SACRAMENTO  z95835     3    2.5  1718  Residential  250000
211  RANCHO_CORDOVA  z95670     3    2.0  1671  Residential  175000

      latitude   longitude  dist
298  38.487409 -121.461413   100
718  38.704554 -121.354753   160
748  38.771917 -121.304439   256
252  38.676658 -121.528128   282
211  38.591477 -121.315340   329
```

Average of nearest points

```python
prediction = nearest_neighbors['price'].mean()
print(prediction)
```

```
280739.2
```

Sample: Visualize new prediction



```
print(prediction)
```

```
280739.2
```

Training, evaluating, and tuning the model

```python
np.random.seed(1)

sacramento_train, sacramento_test = train_test_split(
    sacramento, train_size=0.75
)
```

Note

We are not specifying the stratify argument. The `train_test_split()` function cannot stratify on a quantitative variable

Metric: RMS(P)E

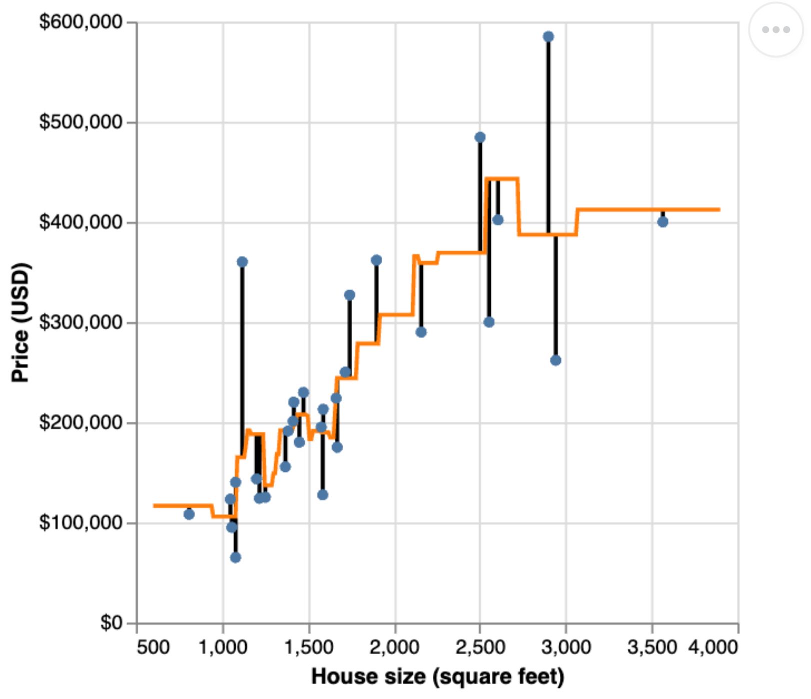
Root Mean Square (Prediction) Error

$$\text{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2}$$

where:

- n is the number of observations,
- y_i is the observed value for the i^{th} observation, and
- $\hat{y_i}$ is the forecasted/predicted value for the i^{th} observation.

Metric: Visualize



RMSPE vs RMSE

Root Mean Square (Prediction) Error

- RMSPE: the error calculated on the non-training dataset
- RMSE: the error calcualted on the training dataset

This notation is a statistics distinction, you will most likely see RMSPE written as RMSE.

Pick the best k: `GridSearchCV()`

We'll use cross validation to try out many different values of k

```python
# import the K-NN regression model
from sklearn.neighbors import KNeighborsRegressor

# preprocess the data, make the pipeline
sacr_preprocessor = make_column_transformer((StandardScaler(), ['sqft']))
sacr_pipeline = make_pipeline(sacr_preprocessor, KNeighborsRegressor())

# create the 5-fold GridSearchCV object
param_grid = {
    'kneighborsregressor__n_neighbors': range(1, 201, 3),
}
sacr_gridsearch = GridSearchCV(
    estimator=sacr_pipeline,
    param_grid=param_grid,
    cv=5,
    scoring='neg_root_mean_squared_error',  # we will deal with this later
)
```

Pick the best k: fit the CV models

```python
# fit the GridSearchCV object
sacr_gridsearch.fit(
    sacramento_train[['sqft']],  # A single-column data frame
    sacramento_train['price'],  # A series
)

# Retrieve the CV scores
sacr_results = pd.DataFrame(sacr_gridsearch.cv_results_)
sacr_results['sem_test_score'] = sacr_results['std_test_score'] / 5 ** (1 / 2)
sacr_results = sacr_results[
    [
        'param_kneighborsregressor__n_neighbors',
        'mean_test_score',
        'sem_test_score',
    ]
].rename(columns={'param_kneighborsregressor__n_neighbors': 'n_neighbors'})
print(sacr_results)
```

```
    n_neighbors  mean_test_score  sem_test_score
0             1   -117365.988307     2715.383001
1             4    -93956.523683     2466.200227
..          ...              ...             ...
65          196    -93671.588088     2473.312705
66          199    -93986.752272     2473.048651

[67 rows x 3 columns]
```

Look at the CV Results

```
print(sacr_results)
```

```
    n_neighbors  mean_test_score  sem_test_score
0             1   -117365.988307     2715.383001
1             4    -93956.523683     2466.200227
..          ...              ...             ...
65          196    -93671.588088     2473.312705
66          199    -93986.752272     2473.048651

[67 rows x 3 columns]
```

- `n_neighbors`: values of K
- `mean_test_score`: RMSPE estimated via cross-validation (it's negative!)
- `sem_test_score`: standard error of our cross-validation RMSPE estimate (how uncertain we are in the mean value)

```
sacr_results['mean_test_score'] = -sacr_results['mean_test_score']
print(sacr_results)
```

```
    n_neighbors  mean_test_score  sem_test_score
0             1    117365.988307     2715.383001
1             4     93956.523683     2466.200227
..          ...              ...             ...
65          196     93671.588088     2473.312705
66          199     93986.752272     2473.048651

[67 rows x 3 columns]
```

Best k

take the *minimum* RMSPE to find the best setting for the number of neighbors

```python
best_k_sacr = sacr_results["n_neighbors"][sacr_results["mean_test_score"].idxmin()]
best_cv_RMSPE = min(sacr_results["mean_test_score"])
```

Best k:

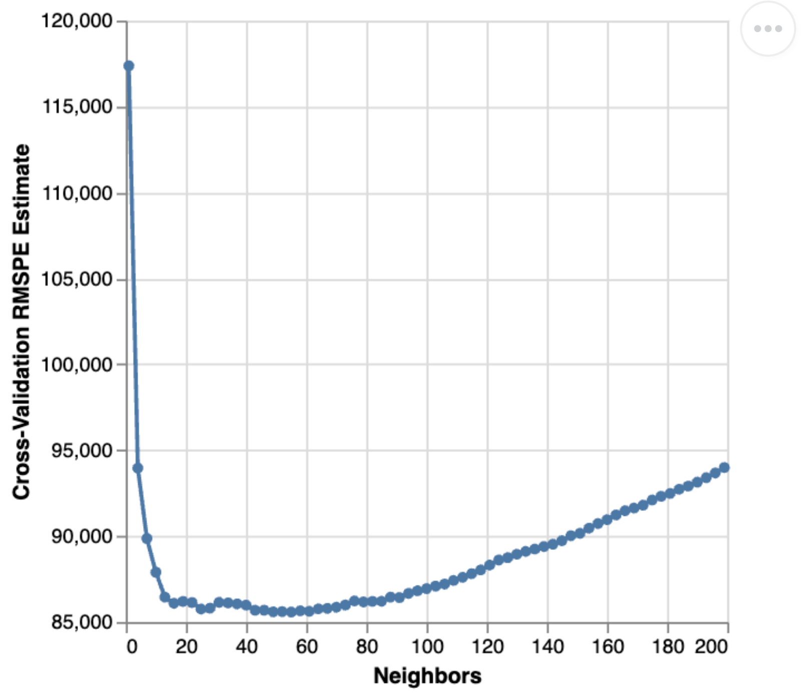
```python
best_k_sacr
```

```
np.int64(55)
```

```python
best_cv_RMSPE
```

```
85578.21347207513
```

Best k: Visualize

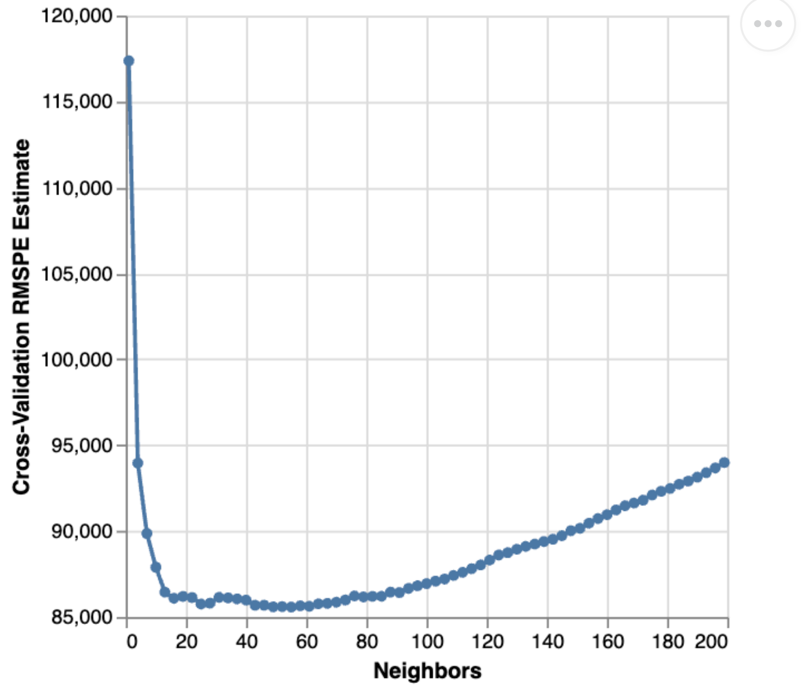


```
sacr_gridsearch.best_params_
```

```
{'kneighborsregressor__n_neighbors': 55}
```

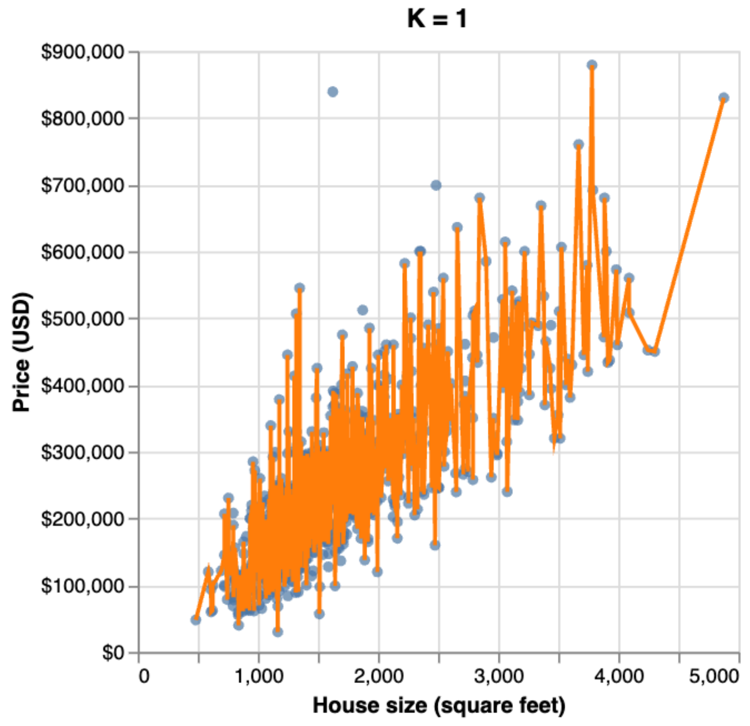
Underfitting and overfitting

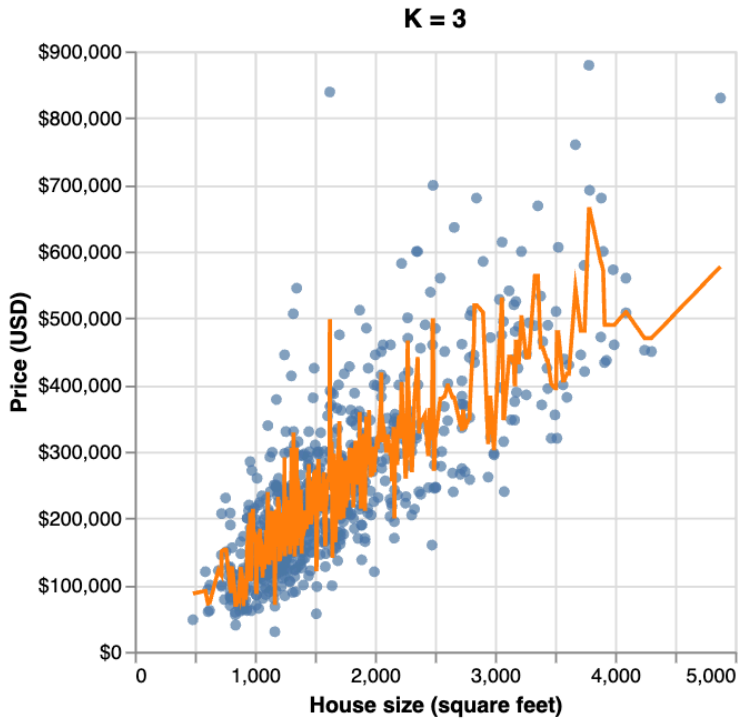
```
sacr_tunek_plot
```

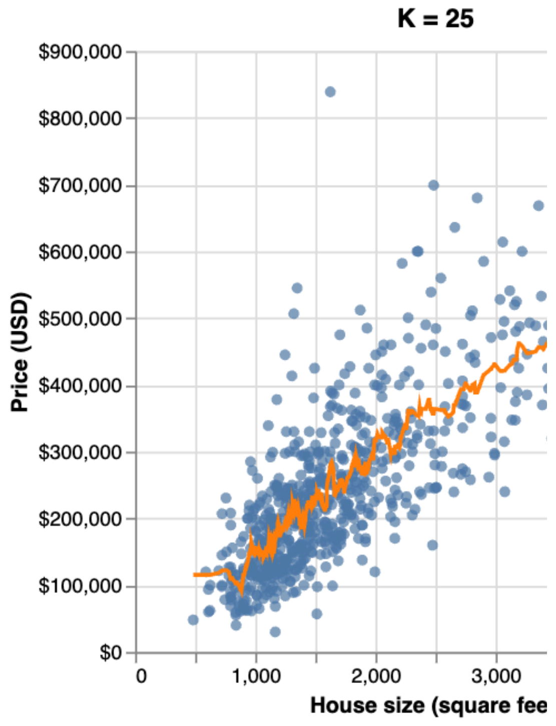


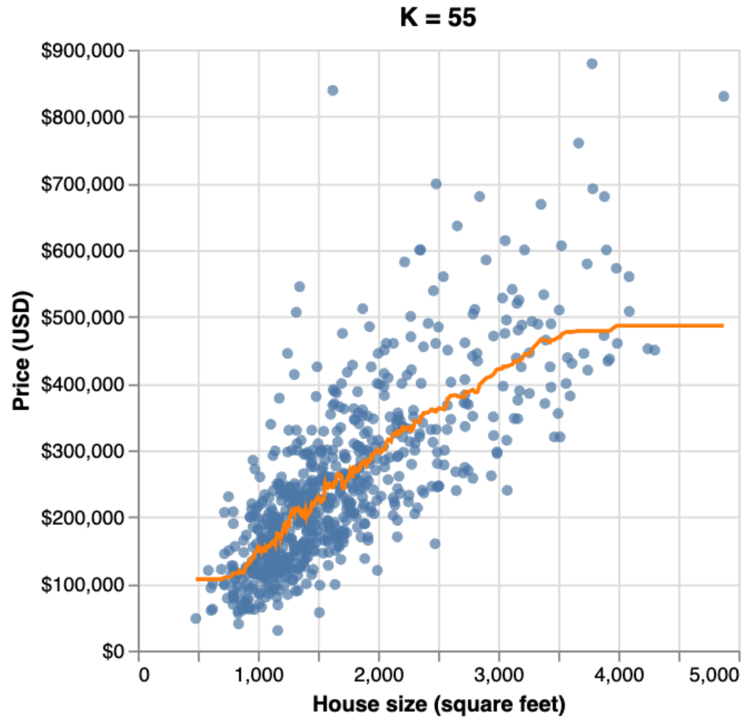
The RMSPE values start to get higher after a certain k value

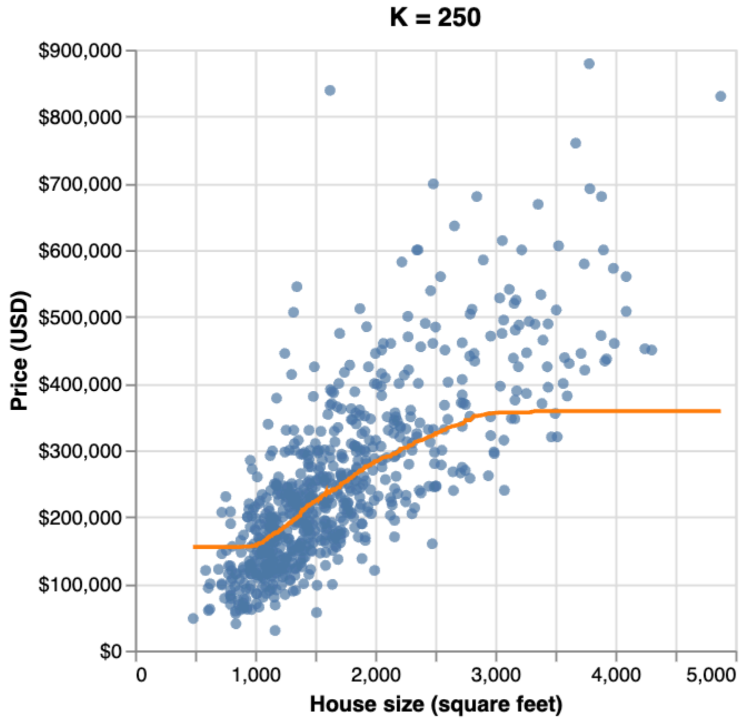
Visualizing different values of k

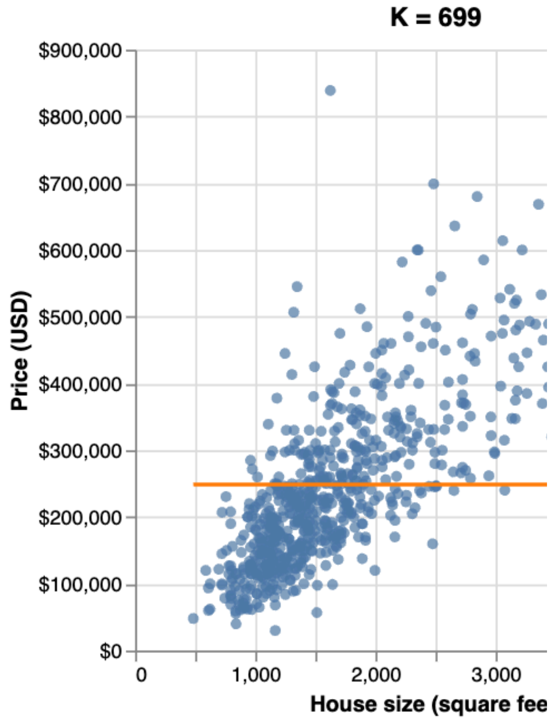

K = 1
Price (USD)
$900,000
$800,000
$700,000
$600,000
$500,000
$400,000
$300,000
$200,000
$100,000
$0
0
1,000
2,000
3,000
4,000
5,000
House size (square feet)


K = 3
Price (USD)
$900,000
$800,000
$700,000
$600,000
$500,000
$400,000
$300,000
$200,000
$100,000
$0
0
1,000
2,000
3,000
4,000
5,000
House size (square feet)


K = 25
Price (USD)
$900,000
$800,000
$700,000
$600,000
$500,000
$400,000
$300,000
$200,000
$100,000
$0
0
1,000
2,000
3,000
House size (square fee


K = 55
Price (USD)
$900,000
$800,000
$700,000
$600,000
$500,000
$400,000
$300,000
$200,000
$100,000
$0
0
1,000
2,000
3,000
4,000
5,000
House size (square feet)


K = 250
Price (USD)
$900,000
$800,000
$700,000
$600,000
$500,000
$400,000
$300,000
$200,000
$100,000
$0
0
1,000
2,000
3,000
4,000
5,000
House size (square feet)


K = 699
Price (USD)
$900,000
$800,000
$700,000
$600,000
$500,000
$400,000
$300,000
$200,000
$100,000
$0
0
1,000
2,000
3,000
House size (square fee

Evaluating on the test set

RMSPE on the test data

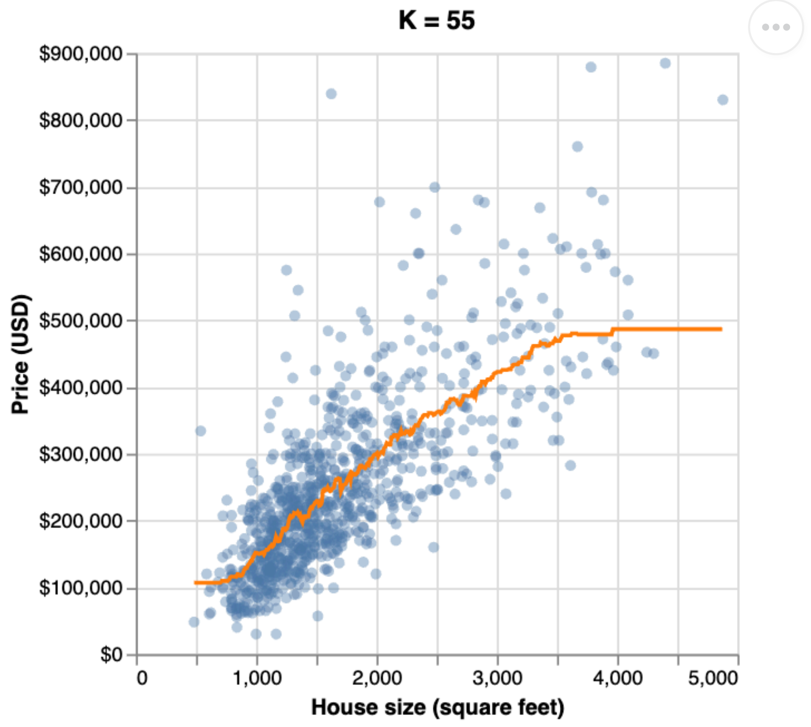
- Retrain the K-NN regression model on the entire training data set using best k

```python
from sklearn.metrics import mean_squared_error

sacramento_test['predicted'] = sacr_gridsearch.predict(sacramento_test)
RMSPE = mean_squared_error(
    y_true=sacramento_test['price'], y_pred=sacramento_test['predicted']
) ** (1 / 2)

RMSPE
```

```
np.float64(87498.86808211416)
```

Final best K model

Predicted values of house price (orange line) for the final K-NN regression model.



Multivariable K-NN regression

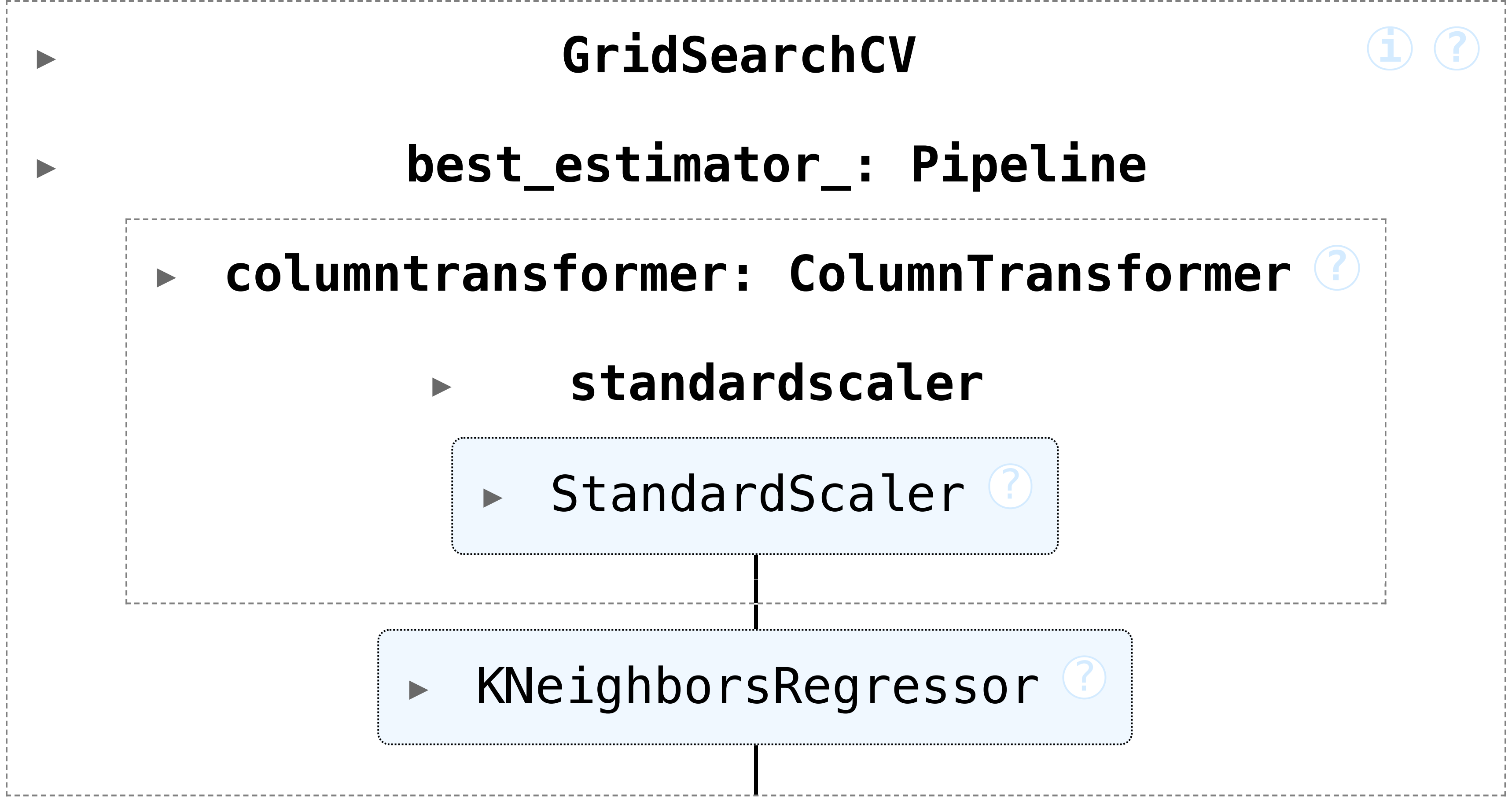
We can use multiple predictors in K-NN regression

Multivariable K-NN regression: Preprocessor

```python
sacr_preprocessor = make_column_transformer(
    (StandardScaler(), ['sqft', 'beds'])
)
sacr_pipeline = make_pipeline(sacr_preprocessor, KNeighborsRegressor())
```

Multivariable K-NN regression: CV

```python
# create the 5-fold GridSearchCV object
param_grid = {
    'kneighborsregressor__n_neighbors': range(1, 50),
}

sacr_gridsearch = GridSearchCV(
    estimator=sacr_pipeline,
    param_grid=param_grid,
    cv=5,
    scoring='neg_root_mean_squared_error',
)

sacr_gridsearch.fit(
    sacramento_train[['sqft', 'beds']], sacramento_train['price']
)
```


GridSearchCV
best_estimator_: Pipeline
columntransformer: ColumnTransformer
standardscaler
StandardScaler
KNeighborsRegressor

Multivariable K-NN regression: Best K

```python
# retrieve the CV scores
sacr_results = pd.DataFrame(sacr_gridsearch.cv_results_)
sacr_results['sem_test_score'] = sacr_results['std_test_score'] / 5 ** (1 / 2)
sacr_results['mean_test_score'] = -sacr_results['mean_test_score']
sacr_results = sacr_results[
    [
        'param_kneighborsregressor__n_neighbors',
        'mean_test_score',
        'sem_test_score',
    ]
].rename(columns={'param_kneighborsregressor__n_neighbors': 'n_neighbors'})

# show only the row of minimum RMSPE
sacr_results.nsmallest(1, 'mean_test_score')
```

	n_neighbors	mean_test_score	sem_test_score
28	29	85156.027067	3376.143313

Multivariable K-NN regression: Best model

```python
best_k_sacr_multi = sacr_results["n_neighbors"][sacr_results["mean_test_score"].idxmin()]
min_rmspe_sacr_multi = min(sacr_results["mean_test_score"])
```

Best K

```python
best_k_sacr_multi
```

```
np.int64(29)
```

Best RMSPE

```python
min_rmspe_sacr_multi
```

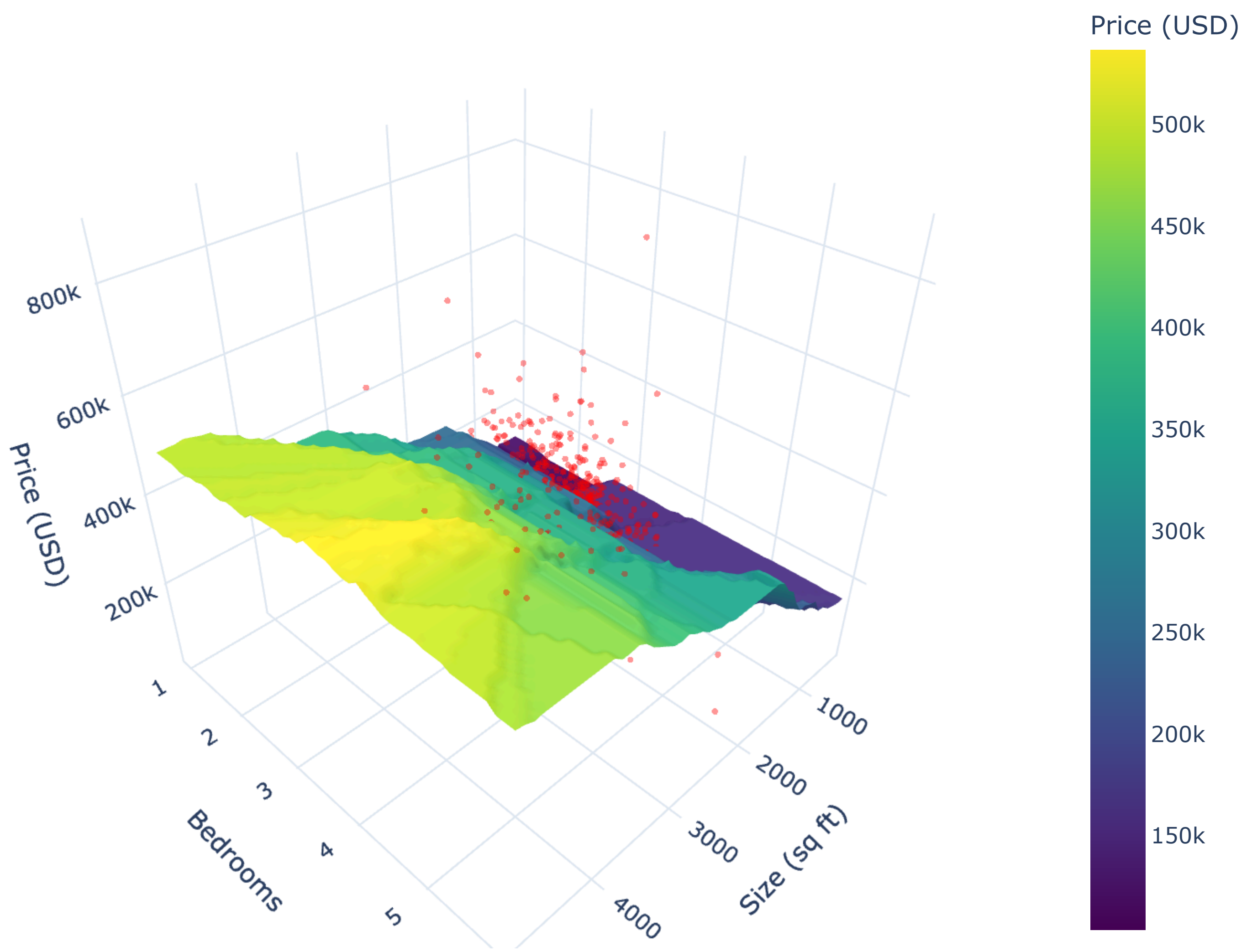
```
85156.02706746716
```

Multivariable K-NN regression: Test data

```python
sacramento_test["predicted"] = sacr_gridsearch.predict(sacramento_test)
RMSPE_mult = mean_squared_error(
    y_true=sacramento_test["price"],
    y_pred=sacramento_test["predicted"]
)**(1/2)

RMSPE_mult
```

```
np.float64(85083.2902421959)
```

Multivariable K-NN regression: Visualize



Strengths and limitations of K-NN regression

Strengths:

- simple, intuitive algorithm
- requires few assumptions about what the data must look like
- works well with non-linear relationships (i.e., if the relationship is not a straight line)

Weaknesses:

- very slow as the training data gets larger
- may not perform well with a large number of predictors
- may not predict well beyond the range of values input in your training data

Linear Regression

- Addresses the limitations from KNN regression
- provides an interpretable mathematical equation that describes the relationship between the predictor and response variables
- Create a straight line of best fit through the training data

Note

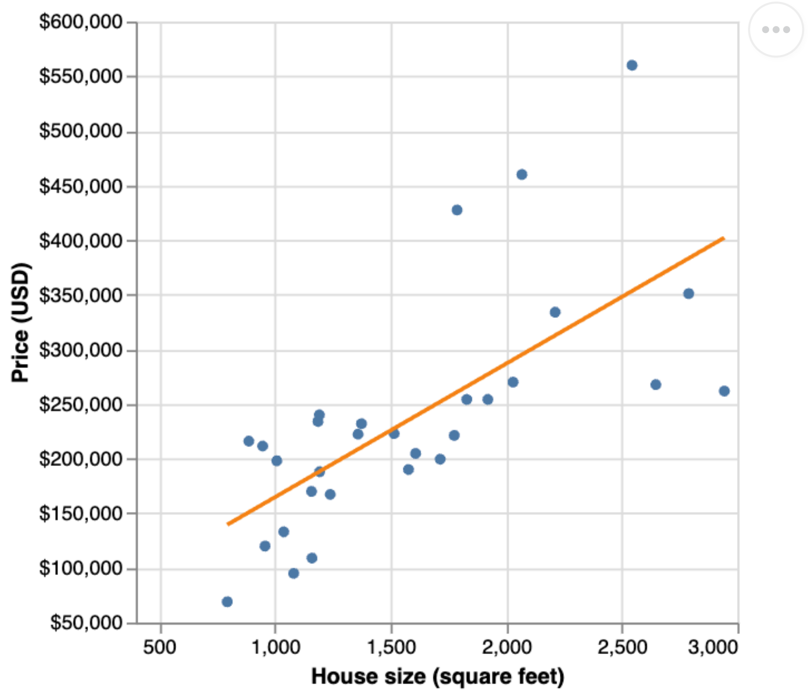
Logistic regression is the linear model we can use for binary classification

Sacramento real estate

```python
import pandas as pd

sacramento = pd.read_csv('data/sacramento.csv')

np.random.seed(42)
small_sacramento = sacramento.sample(n=30)

print(small_sacramento)
```

```
           city     zip  beds  baths  sqft         type   price   latitude  \
829  SACRAMENTO  z95824     3    1.0  1161  Residential  109000  38.511893
70    ELK_GROVE  z95624     4    2.0  1715  Residential  199500  38.440760
..          ...     ...   ...    ...   ...          ...     ...        ...
909  SACRAMENTO  z95828     3    1.0   888  Residential  216021  38.508217
265  SACRAMENTO  z95835     4    3.0  2030  Residential  270000  38.671807

      longitude
829 -121.457676
70  -121.385792
..          ...
909 -121.411207
265 -121.498274

[30 rows x 9 columns]
```

Sacramento real estate: best fit line



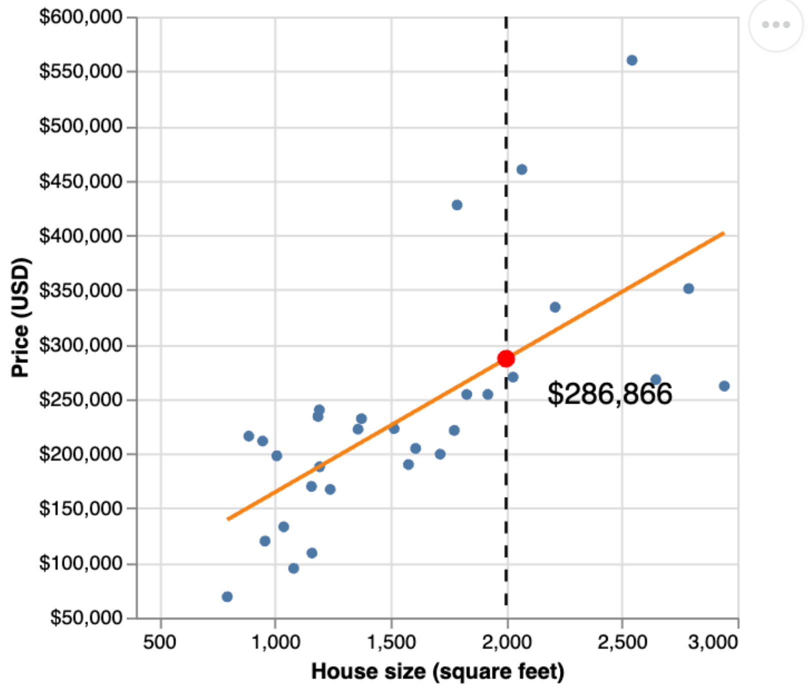
The equation for the straight line is:

$$\text{house sale price} = \beta_0 + \beta_1 \cdot (\text{house size}),$$

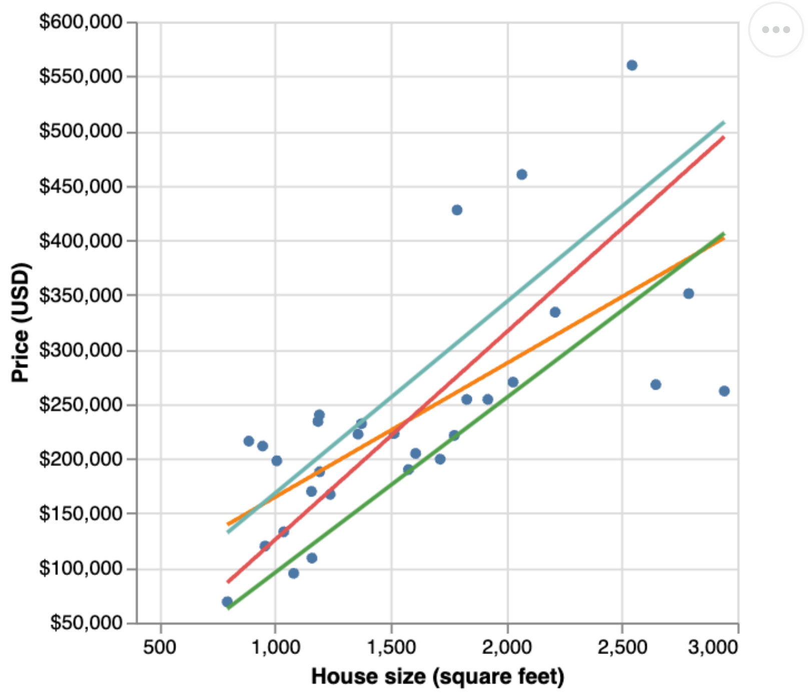
where

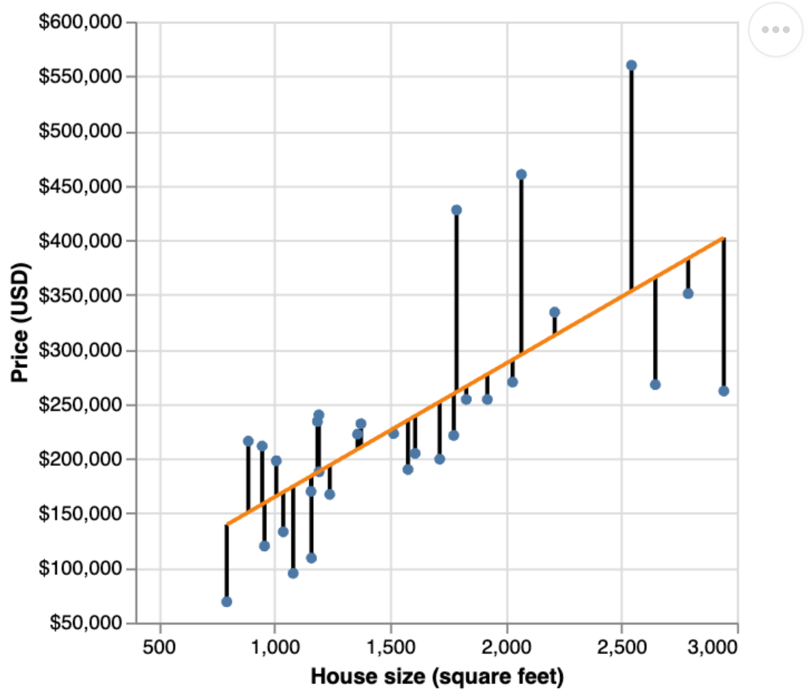
- β_0 is the *vertical intercept* of the line (the price when house size is 0)
- β_1 is the *slope* of the line (how quickly the price increases as you increase house size)

Sacramento real estate: Prediction



What makes the best line?





Linear regression in Python

The `scikit-learn` pattern still applies:

- Create a training and test set
- Instantiate a model
- Fit the model on training
- Use model on testing set

Linear regression: Train test split

```python
import numpy as np
import altair as alt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn import set_config

# Output dataframes instead of arrays
set_config(transform_output='pandas')

np.random.seed(1)

sacramento = pd.read_csv('data/sacramento.csv')

sacramento_train, sacramento_test = train_test_split(
    sacramento, train_size=0.75
)
```

Linear regression: Fit the model

```python
# fit the linear regression model
lm = LinearRegression()
lm.fit(
    sacramento_train[['sqft']],  # A single-column data frame
    sacramento_train['price'],  # A series
)

# make a dataframe containing slope and intercept coefficients
results_df = pd.DataFrame({'slope': [lm.coef_[0]], 'intercept': [lm.intercept_]})
print(results_df)
```

```
        slope     intercept
0  137.285652  15642.309105
```

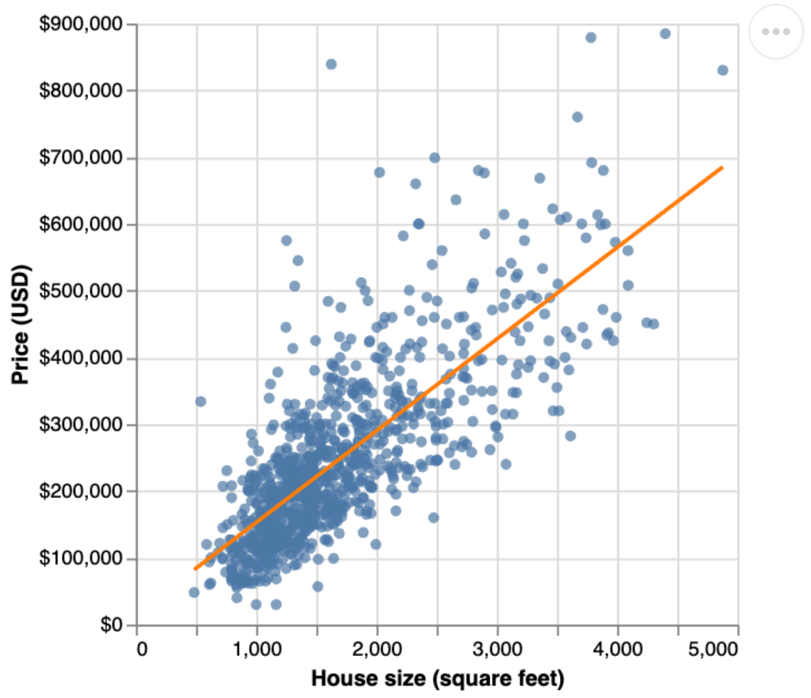
$$\text{house sale price} = 137.29 + 15642.31 \cdot (\text{house size}).$$

Linear regression: Predictions

```python
# make predictions
sacramento_test["predicted"] = lm.predict(sacramento_test[["sqft"]])

# calculate RMSPE
RMSPE = mean_squared_error(
    y_true=sacramento_test["price"],
    y_pred=sacramento_test["predicted"]
)**(1/2)

RMSPE
```

```
np.float64(85376.59691629931)
```

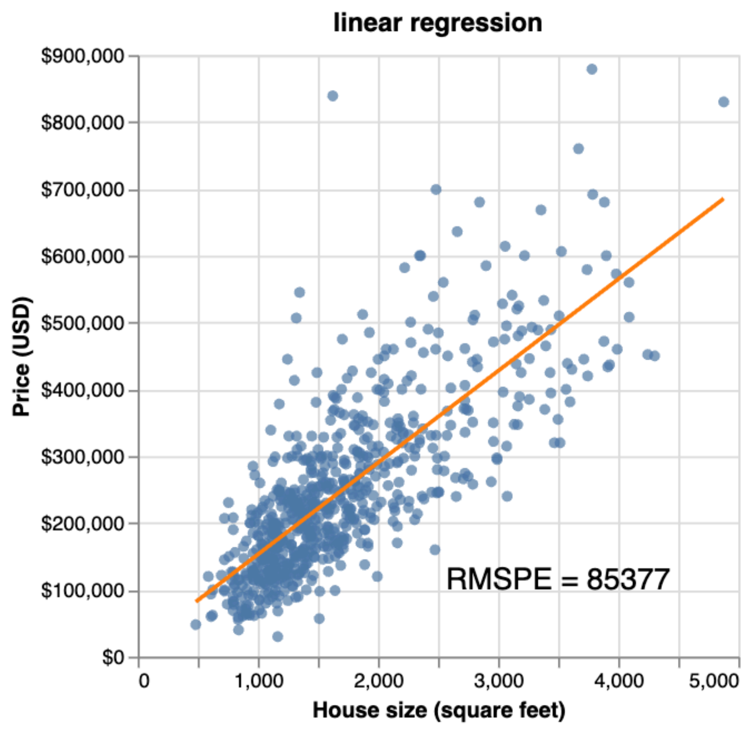
Linear regression: Plot

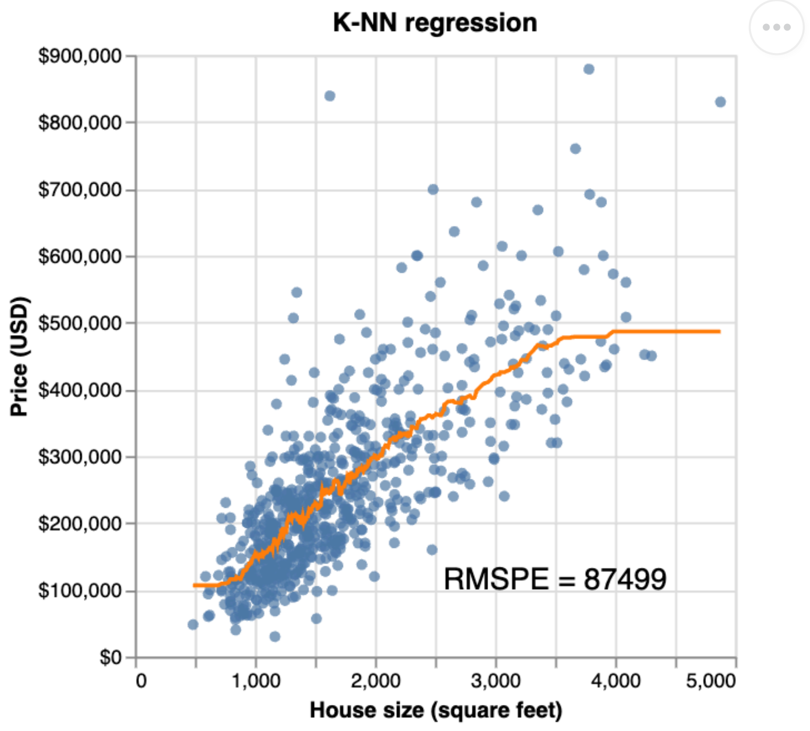


Standarization

- We did not need to standarize like we did for KNN.
- In KNN standarization is mandatory,
- In linear regression, if we standarize, we convert all the units to unit-less standard deviations
- Standarization in linear regression does not change the fit of the model
  - It will change the coefficients

Comparing simple linear and K-NN regression





Multivariable linear regression

More predictor variables!

- More does not always mean better
- We will not cover variable selection in this workshop
- Will talk about categorical predictors later in the workshop

Sacramento real estate: 2 predictors

```
mlm = LinearRegression()
mlm.fit(sacramento_train[['sqft', 'beds']], sacramento_train['price'])

sacramento_test['predicted'] = mlm.predict(sacramento_test[['sqft', 'beds']])
```

Sacramento real estate: Coefficients

Coefficients

```
mlm.coef_
```

```
array([   154.59235377, -20333.43213798])
```

Intercept

```
mlm.intercept_
```

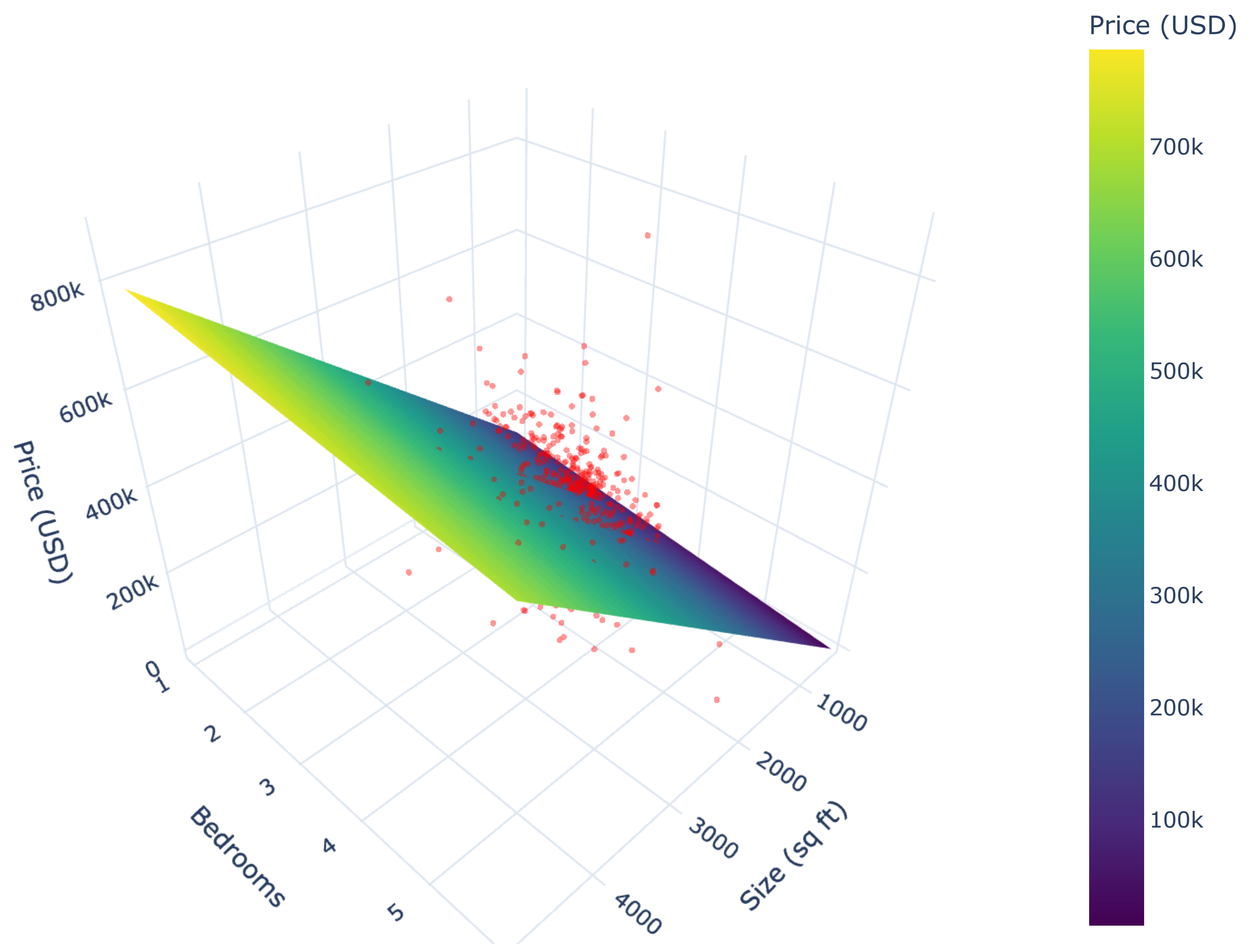
```
np.float64(53180.26906624224)
```

$$\text{house sale price} = \beta_0 + \beta_1 \cdot (\text{house size}) + \beta_2 \cdot (\text{number of bedrooms}),$$

where:

- β_0 is the *vertical intercept* of the hyperplane (the price when both house size and number of bedrooms are 0)
- β_1 is the *slope* for the first predictor (how quickly the price increases as you increase house size)
- β_2 is the *slope* for the second predictor (how quickly the price increases as you increase the number of bedrooms)

More variables make it harder to visualize



Sacramento real estate: mlm rmspe

```python
lm_mult_test_RMSPE = mean_squared_error(
    y_true=sacramento_test['price'], y_pred=sacramento_test['predicted']
) ** (1 / 2)

lm_mult_test_RMSPE
```

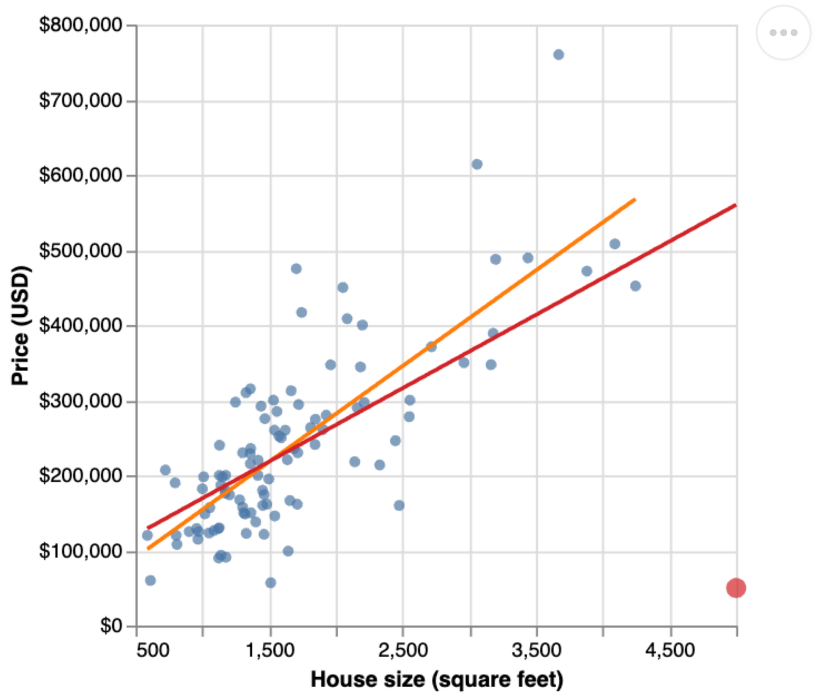
```
np.float64(82331.04630202598)
```

Outliers and Multicollinearity

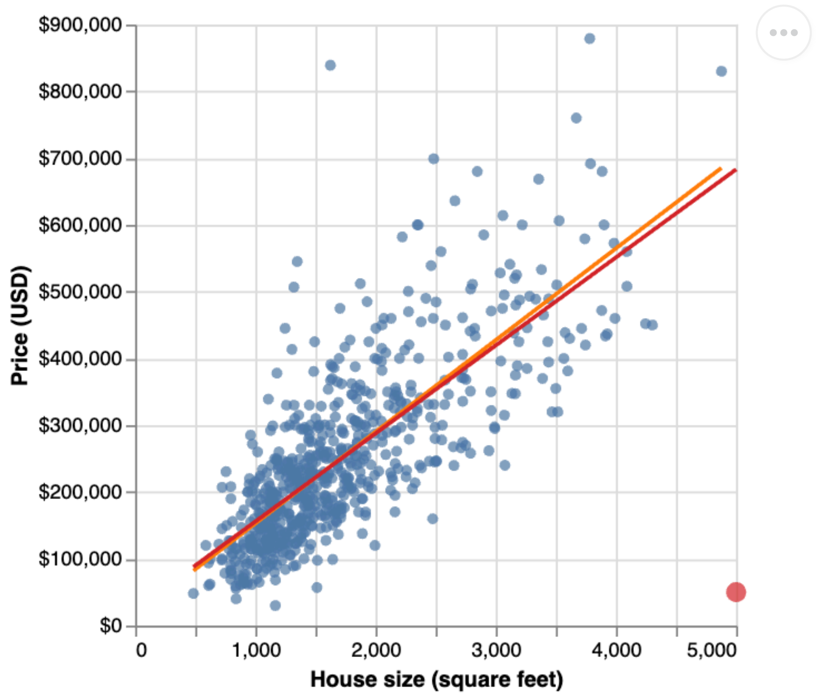
- Outliers: extreme values that can move the best fit line
- Multicollinearity: variables that are highly correlated to one another

Outliers

Subset

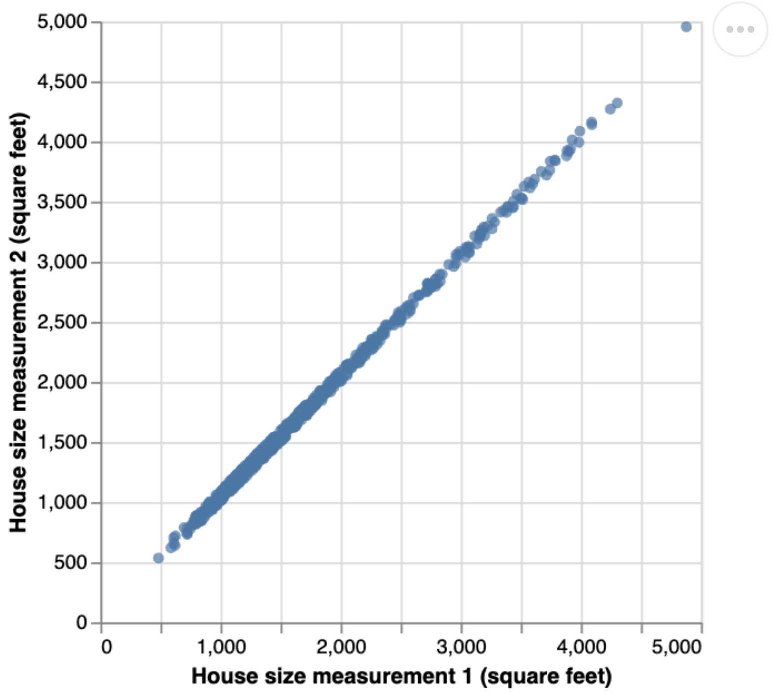


Full data



Multicollinearity

plane of best fit has regression coefficients that are very sensitive to the exact values in the data



- The Regression I: K-nearest neighbors and Regression II: linear regression chapters of Data Science: A First Introduction (Python Edition) by Tiffany Timbers, Trevor Campbell, Melissa Lee, Joel Ostblom, Lindsey Heagy contains all the content presented here with a detailed narrative.
- The `scikit-learn` website is an excellent reference for more details on, and advanced usage of, the functions and packages in this lesson. Aside from that, it also offers many useful tutorials to get you started.
- *An Introduction to Statistical Learning* by Gareth James Daniela Witten Trevor Hastie, and Robert Tibshirani provides a great next stop in the process of learning about classification. Chapter 3 discusses lienar regression in more depth. As well as how it comares to K-nearest neighbors.

References

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1):21–27, 1967.

Evelyn Fix and Joseph Hodges. Discriminatory analysis. nonparametric discrimination: consistency properties. Technical Report, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.