What is regression?

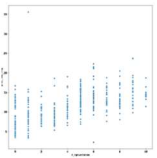
- Statistical models to explore the relationship between a response variable and some explanatory variables.
- Given values of explanatory variables, you can predict the values of the response variable.
- **Response variable** (a.k.a. dependent variable) The variable that you want to predict.
- **Explanatory variables** (a.k.a. independent variables) The variables that explain how the response variable will change.

Visualizing two numeric variables
Before you can run any statistical models, it's usually a good idea to visualize your dataset. Here, you'll look at the relationship between house price per area and the number of nearby convenience stores using the Taiwan real estate dataset.

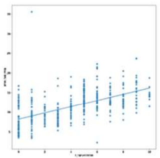
One challenge in this dataset is that the number of convenience stores contains integer data, causing points to overlap. To solve this, you will make the points transparent.

taiwan_real_estate is available from Taiwan_real_estate2.csv

- Import the seaborn and matplotlib packages
- Using taiwan_real_estate, draw a scatter plot of "price_twd_msq" (y-axis) versus "n_convenience" (x-axis).



- Draw a trend line calculated using linear regression. Omit the confidence interval ribbon. Note: The scatter_kws argument in regplot, scatter_kws={'alpha': 0.5}, makes the data points 50% transparent.



Transforming variables

- Sometimes, the relationship between the explanatory variable and the response variable may not be a straight line. To fit a linear regression model, you may need to transform the explanatory variable or the response variable, or both of them.

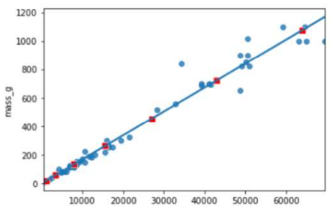
Modeling mass vs. length cubed

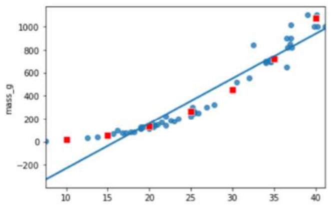
```
perch["length_cm_cubed"] = perch["length_cm"] ** 3

mdl_perch = ols("mass_g ~ length_cm_cubed", data=perch).fit()
mdl_perch.params
```

```
Intercept          -0.117478
length_cm_cubed     0.016796
dtype: float64
```

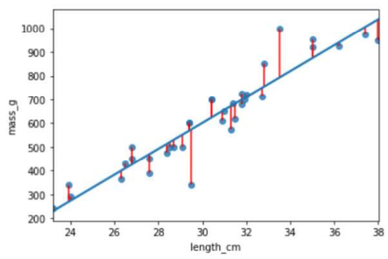
```
fig = plt.figure()
sns.regplot(x="length_cm_cubed", y="mass_g",
            data=perch, ci=None)
sns.scatterplot(data=prediction_data,
                x="length_cm_cubed", y="mass_g",
                color="red", marker="s")
```

```
fig = plt.figure()
sns.regplot(x="length_cm", y="mass_g",
            data=perch, ci=None)
sns.scatterplot(data=prediction_data,
                x="length_cm", y="mass_g",
                color="red", marker="s")
```





Residual standard error (RSE)



- A "typical" difference between a prediction and an observed response
- It has the same unit as the response variable.
- MSE = RSE^2

Using scikit-learn to fit a classifier

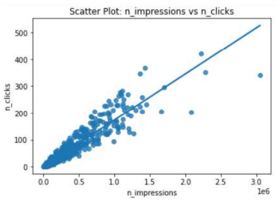
```
from sklearn.neighbors import KNeighborsClassifier
X = churn_df[["total_day_charge", "total_eve_charge"]].values
y = churn_df["churn"].values
print(X.shape, y.shape)
```

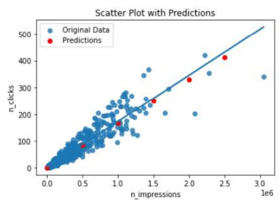
```
(3333, 2), (3333,)
```

```
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X, y)
```

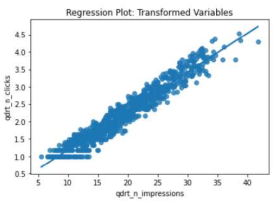
```
#Lut Lat Aung 6511163

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.formula.api import ols

ad_conversion = pd.read_csv("ad_conversion.csv")

sns.regplot(x="n_impressions", y="n_clicks", data=ad_conversion, ci=None)
plt.title("Scatter Plot: n_impressions vs n_clicks")
plt.xlabel("n_impressions")
plt.ylabel("n_clicks")
plt.show()

ad_conversion["qdrt_n_impressions"] = ad_conversion["n_impressions"] ** 0.25
ad_conversion["qdrt_n_clicks"] = ad_conversion["n_clicks"] ** 0.25
sns.regplot(x="qdrt_n_impressions", y="qdrt_n_clicks", data=ad_conversion, ci=None)
plt.title("Regression Plot: Transformed Variables")
plt.xlabel("qdrt_n_impressions")
plt.ylabel("qdrt_n_clicks")
plt.show()

# Do the points track the line more closely? Yes.

mdl_click_vs_impression = ols("qdrt_n_clicks ~ qdrt_n_impressions", data=ad_conversion).fit()
print("Model Parameters:")
print(mdl_click_vs_impression.params)

explanatory_data = pd.DataFrame({
    "qdrt_n_impressions": np.arange(0, 3000000, 500000) ** 0.25,
    "n_impressions": np.arange(0, 3000000, 500000)
})
prediction_data = explanatory_data.assign(
    qdrt_n_clicks=mdl_click_vs_impression.predict(explanatory_data)
)
```

```
print("Prediction Data:")
print(prediction_data)

# Back transformation:
prediction_data["n_clicks"] = prediction_data["qdrt_n_clicks"] ** 4
sns.regplot(x="n_impressions", y="n_clicks", data=ad_conversion, ci=None, label="Original Data")
plt.scatter(
    x=prediction_data["n_impressions"],
    y=prediction_data["n_clicks"],
    color="red",
    label="Predictions",
)
plt.title("Scatter Plot with Predictions")
plt.xlabel("n_impressions")
plt.ylabel("n_clicks")
plt.legend()
plt.show()


r_squared = mdl_click_vs_impression.rsquared
print("R-squared:", r_squared)
residuals = ad_conversion["qdrt_n_clicks"] - mdl_click_vs_impression.fittedvalues
rse = np.sqrt(np.sum(residuals ** 2) / (len(ad_conversion) - 2))
print("RSE:", rse)

# Interpret the result:
# - A higher R-squared value indicates a better fit, with 1.0 being a perfect fit.
# - RSE measures the standard deviation of the residuals, indicating how well the model fits the data.
# - A lower RSE suggests a better fit, as it means the model's predictions are closer to the actual values.
```





```
R-squared: 0.9445272817143905
RSE: 0.19690640896875727
```



```
Model Parameters:
Intercept             0.071748
qdrt_n_impressions    0.111533
dtype: float64
Prediction Data:
   qdrt_n_impressions  n_impressions  qdrt_n_clicks
0            0.000000              0       0.071748
```

scikit-learn syntax

```
from sklearn.module import Model
model = Model()
model.fit(X, y)
predictions = model.predict(X_new)
print(predictions)
```

```
array([0, 0, 0, 0, 1, 0])
```

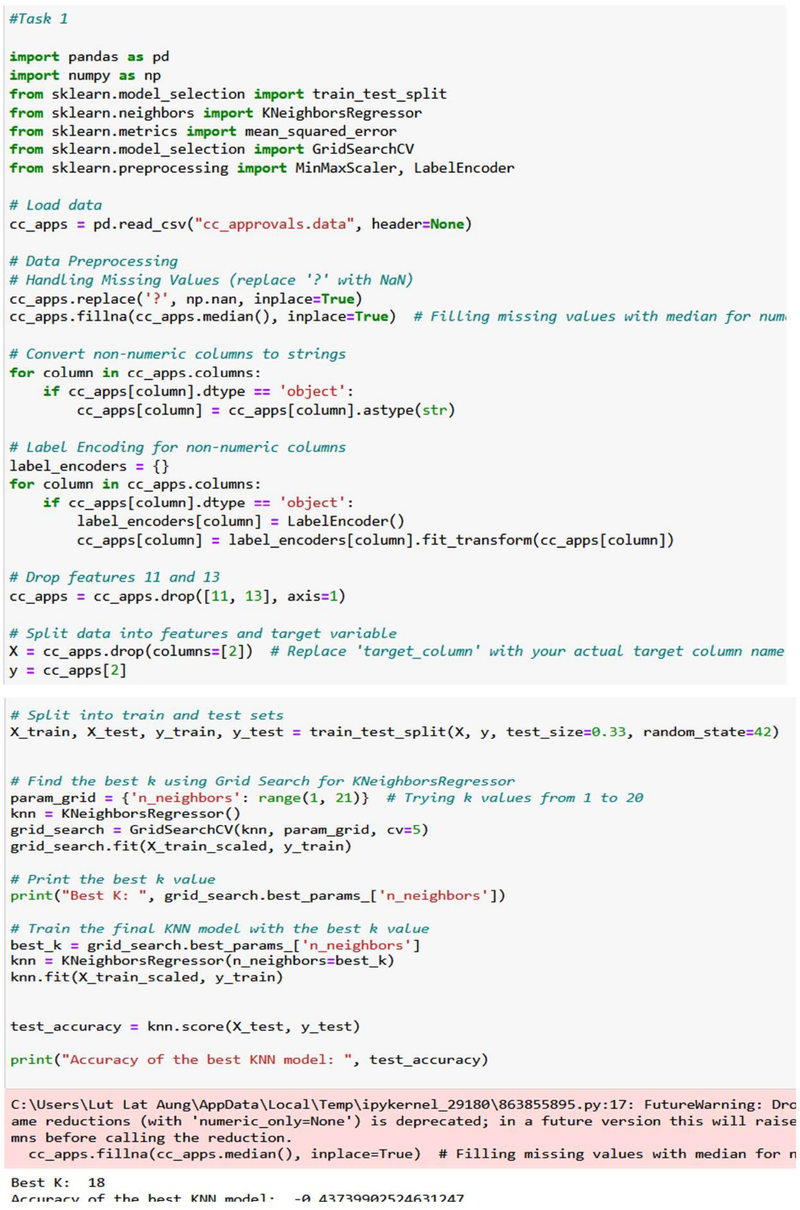
Classifying labels of unseen data

1. Build a model
2. Model learns from the labeled data we pass to it
3. Pass unlabeled data to the model as input
4. Model predicts the labels of the unseen data

- Labeled data = training data

k-Nearest Neighbors

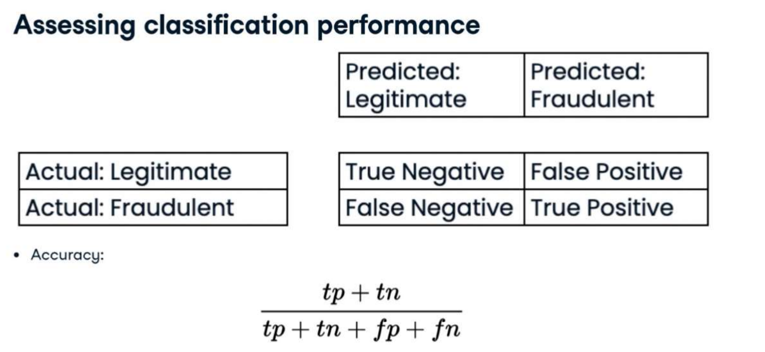
- Predict the label of a data point by
  - Looking at the `k` closest labeled data points
  - Taking a majority vote



```
#Task 1

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler, LabelEncoder

# Load data
cc_apps = pd.read_csv("cc_approvals.data", header=None)

# Data Preprocessing
# Handling Missing Values (replace '?' with NaN)
cc_apps.replace('?', np.nan, inplace=True)
cc_apps.fillna(cc_apps.median(), inplace=True)  # Filling missing values with median for num

# Convert non-numeric columns to strings
for column in cc_apps.columns:
    if cc_apps[column].dtype == 'object':
        cc_apps[column] = cc_apps[column].astype(str)

# Label Encoding for non-numeric columns
label_encoders = {}
for column in cc_apps.columns:
    if cc_apps[column].dtype == 'object':
        label_encoders[column] = LabelEncoder()
        cc_apps[column] = label_encoders[column].fit_transform(cc_apps[column])

# Drop features 11 and 13
cc_apps = cc_apps.drop([11, 13], axis=1)

# Split data into features and target variable
X = cc_apps.drop(columns=[2])  # Replace 'target_column' with your actual target column name
y = cc_apps[2]
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)


# Find the best k using Grid Search for KNeighborsRegressor
param_grid = {'n_neighbors': range(1, 21)}  # Trying k values from 1 to 20
knn = KNeighborsRegressor()
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

# Print the best k value
print("Best K: ", grid_search.best_params_['n_neighbors'])

# Train the final KNN model with the best k value
best_k = grid_search.best_params_['n_neighbors']
knn = KNeighborsRegressor(n_neighbors=best_k)
knn.fit(X_train_scaled, y_train)


test_accuracy = knn.score(X_test, y_test)

print("Accuracy of the best KNN model: ", test_accuracy)
```

```
C:\Users\Lut Lat Aung\AppData\Local\Temp\ipykernel_29180\863855895.py:17: FutureWarning: Dro
ame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise
mns before calling the reduction.
  cc_apps.fillna(cc_apps.median(), inplace=True)  # Filling missing values with median for n

Best K:  18
Accuracy of the best KNN model:  -0.43739902524631247
```

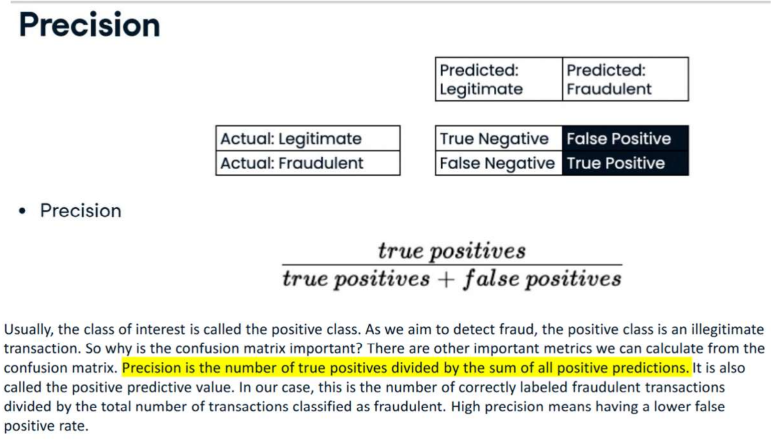


Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

- Accuracy:

$$\frac{tp + tn}{tp + tn + fp + fn}$$



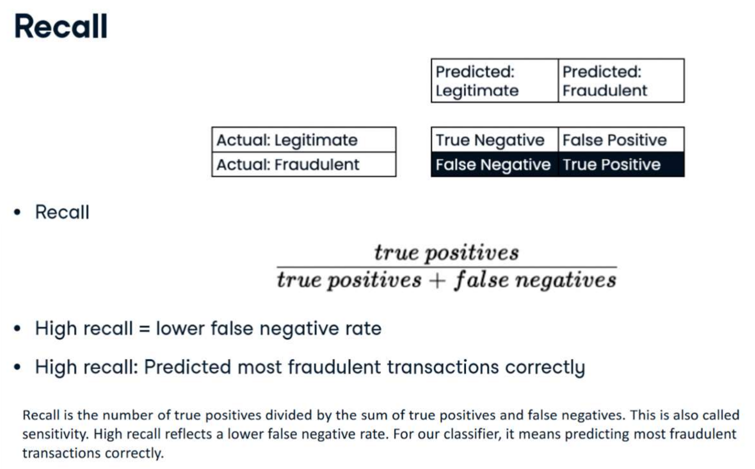
Precision

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

- Precision

$$\frac{true\ positives}{true\ positives + false\ positives}$$

Usually, the class of interest is called the positive class. As we aim to detect fraud, the positive class is an illegitimate transaction. So why is the confusion matrix important? There are other important metrics we can calculate from the confusion matrix. Precision is the number of true positives divided by the sum of all positive predictions. It is also called the positive predictive value. In our case, this is the number of correctly labeled fraudulent transactions divided by the total number of transactions classified as fraudulent. High precision means having a lower false positive rate.



Recall

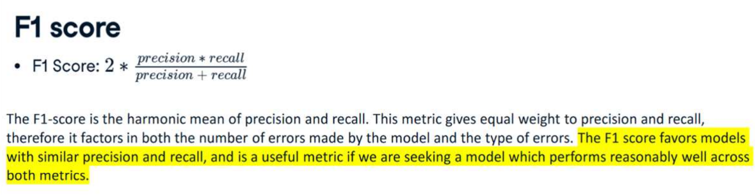
	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

- Recall

$$\frac{true\ positives}{true\ positives + false\ negatives}$$

- High recall = lower false negative rate
- High recall: Predicted most fraudulent transactions correctly

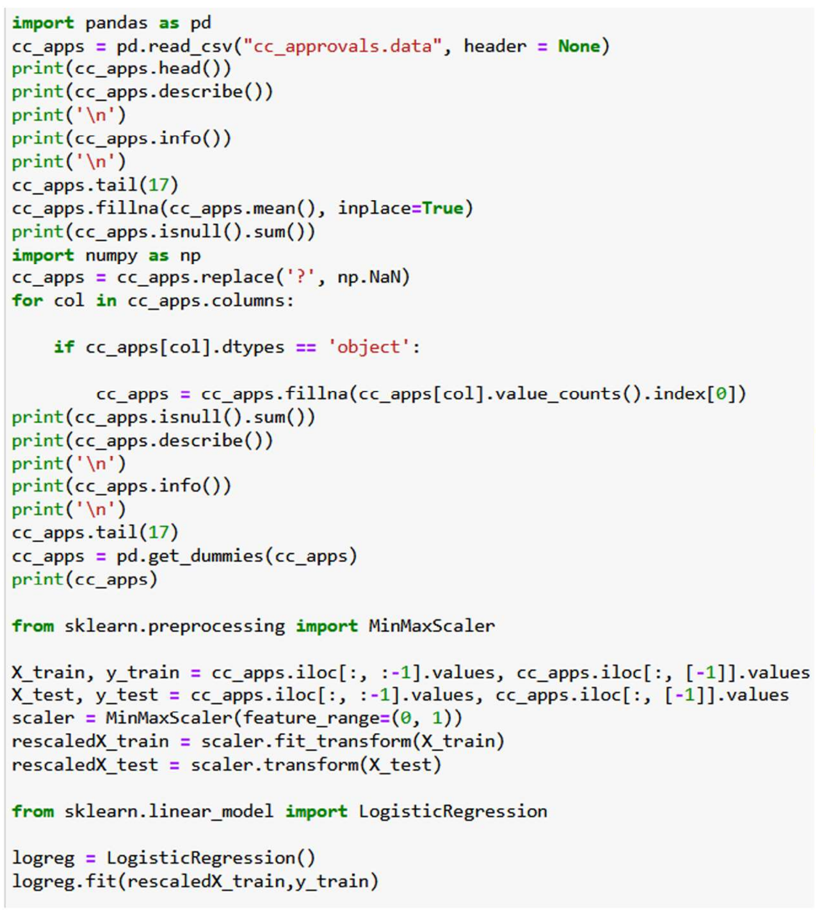
Recall is the number of true positives divided by the sum of true positives and false negatives. This is also called sensitivity. High recall reflects a lower false negative rate. For our classifier, it means predicting most fraudulent transactions correctly.



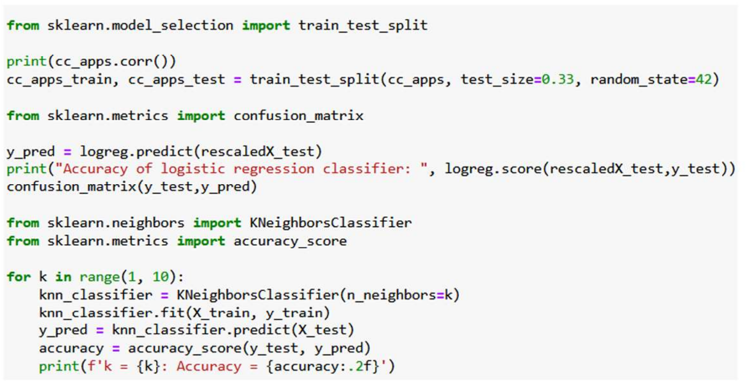
F1 score

- F1 Score: $2 * \frac{precision * recall}{precision + recall}$

The F1-score is the harmonic mean of precision and recall. This metric gives equal weight to precision and recall, therefore it factors in both the number of errors made by the model and the type of errors. The F1 score favors models with similar precision and recall, and is a useful metric if we are seeking a model which performs reasonably well across both metrics.



```
import pandas as pd
cc_apps = pd.read_csv("cc_approvals.data", header = None)
print(cc_apps.head())
print(cc_apps.describe())
print('\n')
print(cc_apps.info())
print('\n')
cc_apps.tail(17)
cc_apps.fillna(cc_apps.mean(), inplace=True)
print(cc_apps.isnull().sum())
import numpy as np
cc_apps = cc_apps.replace('?', np.NaN)
for col in cc_apps.columns:

    if cc_apps[col].dtypes == 'object':

        cc_apps = cc_apps.fillna(cc_apps[col].value_counts().index[0])
print(cc_apps.isnull().sum())
print(cc_apps.describe())
print('\n')
print(cc_apps.info())
print('\n')
cc_apps.tail(17)
cc_apps = pd.get_dummies(cc_apps)
print(cc_apps)

from sklearn.preprocessing import MinMaxScaler

X_train, y_train = cc_apps.iloc[:, :-1].values, cc_apps.iloc[:, [-1]].values
X_test, y_test = cc_apps.iloc[:, :-1].values, cc_apps.iloc[:, [-1]].values
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(rescaledX_train,y_train)
```



```
from sklearn.model_selection import train_test_split

print(cc_apps.corr())
cc_apps_train, cc_apps_test = train_test_split(cc_apps, test_size=0.33, random_state=42)

from sklearn.metrics import confusion_matrix

y_pred = logreg.predict(rescaledX_test)
print("Accuracy of logistic regression classifier: ", logreg.score(rescaledX_test,y_test))
confusion_matrix(y_test,y_pred)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

for k in range(1, 10):
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'k = {k}: Accuracy = {accuracy:.2f}')
```

k-Nearest Neighbors: Fit

In this exercise, you will build your first classification model using the churn_df dataset, can be loaded from churn_df.csv.

The features to use will be "account_length" and "customer_service_calls". The target, "churn", needs to be a single column with the same number of observations as the feature data.

You will convert the features and the target variable into NumPy arrays, create an instance of a KNN classifier, and then fit it to the data.

Instructions

- Import KNeighborsClassifier from sklearn.neighbors.
- Create an array called X containing values from the "account_length" and "customer_service_calls" columns, and an array called y for the values of the "churn" column.
- Instantiate a KNeighborsClassifier called knn with 6 neighbors.
- Fit the classifier to the data using the .fit() method.

```
X = churn_df[['total_day_charge', 'total_eve_charge']].values
y = churn_df['churn'].values
print(X.shape, y.shape)
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X,y)
X_new = np.array([[56.8, 17.5],
                  [24.4, 24.1],
                  [50.1, 10.9]])
print(X_new.shape)
predictions = knn.predict(X_new)
print(f'Predictions: {predictions}')
```

k-Nearest Neighbors: Predict

Now you have fit a KNN classifier, you can use it to predict the label of new data points. All available data was used for training, however, fortunately, there are new observations available, X_new.

The model knn, which you created and fit the data in the last exercise, will be used. You will use your classifier to predict the labels of a set of new data points:

X_new = np.array([[30.0, 17.5],
[107.0, 24.1],
[213.0, 10.9]])

```
X = churn_df[['total_day_charge', 'total_eve_charge']].values
y = churn_df['churn'].values
print(X.shape, y.shape)
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X,y)
X_new = np.array([[56.8, 17.5],
                  [24.4, 24.1],
                  [50.1, 10.9]])
print(X_new.shape)
predictions = knn.predict(X_new)
print(f'Predictions: {predictions}')
```

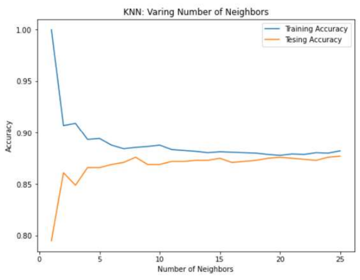
Instructions

- Create y_pred by predicting the target values of the unseen features X_new.
- Print the predicted labels for the set of predictions.

```
Predictions: [0 1 0]
```

```
import matplotlib.pyplot as plt


train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1,26)
print(neighbors)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors = neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

#print(train_accuracies.values())
print(test_accuracies.values())
my_train = list(train_accuracies.values())
my_test = list(test_accuracies.values())

plt.figure(figsize=(8,6))
plt.title("KNN: Varing Number of Neighbors")
plt.plot(neighbors, my_train, label="Training Accuracy")
plt.plot(neighbors, my_test, label="Tesing Accuracy")
plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.show()
```

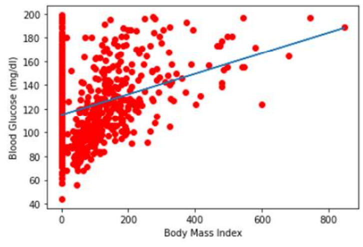
```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25]
dict_values([0.795, 0.861, 0.849, 0.866, 0.866, 0.869, 0.871, 0.876, 0.869, 0.869, 0.872, 0.872, 0.873, 0.873, 0.875, 0.871, 0.
872, 0.873, 0.875, 0.876, 0.875, 0.874, 0.873, 0.876, 0.877])
```



```
import pandas as pd
import numpy as np
sales_df = pd.read_csv('sales_df.csv')

X = sales_df['radio'].values
y = sales_df['sales'].values

X = X.reshape(-1, 1)
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
Shape of X: (4546, 1)
Shape of y: (4546,)
```

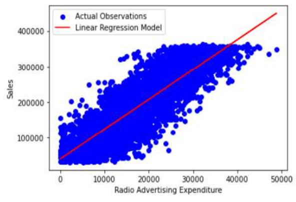
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_bmi, y)
predictions = reg.predict(X_bmi)
plt.scatter(X_bmi, y, color="Red")
plt.plot(X_bmi, predictions)
plt.ylabel("Blood Glucose (mg/dl)")
plt.xlabel("Body Mass Index")
plt.show()
```



```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)

predictions = model.predict(X)

print("Five prediction values:", predictions[:5])
```

```
Five prediction values: [ 95491.17119147 117829.51
 111137.28167129]
```

```
import matplotlib.pyplot as plt
plt.scatter(X, y, color='blue', label='Actual Observations')

plt.plot(X, predictions, color='red', label='Linear Regression Model')

plt.xlabel('Radio Advertising Expenditure')
plt.ylabel('Sales')
plt.legend(loc='best')
plt.show()
```



```
from sklearn.model_selection import train_test_split

print(cc_apps.corr())

#Drop the features 11 and 13
cc_apps = cc_apps.drop([11, 13], axis=1)

# Split into train and test sets
cc_apps_train, cc_apps_test = train_test_split(cc_apps, test_size=0.33, random_state=42)
```

```
           2         7        10        14
2   1.000000  0.298902  0.271207  0.123121
7   0.298902  1.000000  0.322330  0.051345
10  0.271207  0.322330  1.000000  0.063692
14  0.123121  0.051345  0.063692  1.000000
```

```
# Import numpy
import numpy as np

# Replace the '?'s with NaN in the train and test sets
cc_apps_train = cc_apps_train.replace('?', np.NaN)
cc_apps_test = cc_apps_test.replace('?', np.NaN)
```

```
# Impute the missing values with mean imputation
cc_apps_train.fillna(cc_apps_train.mean(), inplace=True)
cc_apps_test.fillna(cc_apps_train.mean(), inplace=True)

# Count the number of NaNs in the datasets and print the counts to verify
print(cc_apps_train.isnull().sum())
print(cc_apps_test.isnull().sum())
```

```
for col in cc_apps_train.columns: # Iterate over each column of cc_apps_train

    if cc_apps_train[col].dtypes == 'object': # Check if the column is of object type
        # Impute with the most frequent value
        # The value_counts() function returns a Series that contain counts of unique val
        # descending order so that its first element will be the most frequently-occurre
        cc_apps_train = cc_apps_train.fillna(cc_apps_train[col].value_counts().index[0])
        cc_apps_test = cc_apps_test.fillna(cc_apps_train[col].value_counts().index[0])

# Count the number of NaNs in the dataset and print the counts to verify
print(cc_apps_train.isnull().sum())
print(cc_apps_test.isnull().sum())
# At this point, there is no missing values.
```

```
# Convert the categorical features in the train and test sets independently
print(cc_apps_train)
cc_apps_train = pd.get_dummies(cc_apps_train)
cc_apps_test = pd.get_dummies(cc_apps_test)
print(cc_apps_train)
# Reindex the columns of the test set aligning with the train set
cc_apps_test = cc_apps_test.reindex(columns=cc_apps_train.columns, fill_value=0)
```

```
# Import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

# Segregate features and labels into separate variables
X_train, y_train = cc_apps_train.iloc[:, :-1].values, cc_apps_train.iloc[:, [-1]].values
X_test, y_test = cc_apps_test.iloc[:, :-1].values, cc_apps_test.iloc[:, [-1]].values

# Instantiate MinMaxScaler and use it to rescale X_train and X_test
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)
```

```
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Instantiate a LogisticRegression classifier with default parameter values
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(rescaledX_train,y_train)
```

```
# Import confusion_matrix
from sklearn.metrics import confusion_matrix

# Use logreg to predict instances from the test set and store it
y_pred = logreg.predict(rescaledX_test)

# Get the accuracy score of logreg model and print it
print("Accuracy of logistic regression classifier: ", logreg.score(rescaledX_test,y_test))

# Print the confusion matrix of the logreg model
confusion_matrix(y_test,y_pred)
```

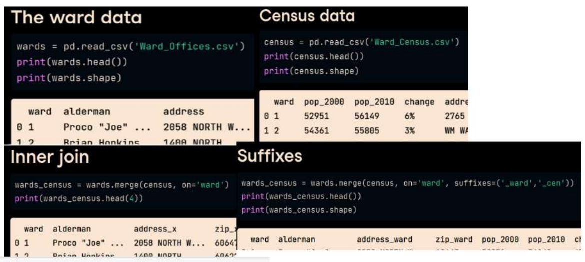
```
Accuracy of logistic regression classifier:  1.0

array([[103,   0],
       [  0, 125]], dtype=int64)
```

Joining Data

The pandas package is a powerful tool for manipulating and transforming data in Python. However, when working on an analysis, the data needed could be in multiple tables. This worksheet will focus on the vital skill of merging tables together.

As an example, we are considering wards data of Chicago city.

The city of Chicago is divided into fifty local neighborhoods called wards. We have a table with data about the local government offices in each ward. In this example, we want to merge the local government data with census data about the population of each ward.



The ward data

```
wards = pd.read_csv('Ward_Offices.csv')
print(wards.head())
print(wards.shape)
```

```
   ward  alderman          address
0  1     Proco "Joe" ...   2058 NORTH W...
1  2     Brian Hopkins     1400 NORTH
```

Census data

```
census = pd.read_csv('Ward_Census.csv')
print(census.head())
print(census.shape)
```

```
   ward  pop_2000  pop_2010  change  addre
0  1     52951     56149     6%      2765
1  2     54361     55805     3%      WM WA
```

Inner join

```
wards_census = wards.merge(census, on='ward')
print(wards_census.head(4))
```

```
   ward  alderman          address_x         zip_
0  1     Proco "Joe" ...   2058 NORTH W...   6064
1  2     Brian Hopkins     1400 NORTH        6062
```

Suffixes

```
wards_census = wards.merge(census, on='ward', suffixes=('_ward','_cen'))
print(wards_census.head())
print(wards_census.shape)
```

```
   ward  alderman          address_ward      zip_ward  pop_2000  pop_2010  ch
```

```
#Ex 1

taxi_owners = pd.read_pickle("taxi_owners.p")
taxi_vehicles = pd.read_pickle("taxi_vehicles.p")
taxi_own_veh = taxi_owners.merge(taxi_vehicles, on = "vid")
print(taxi_own_veh["fuel_type"].value_counts())
print()

print("The highest is : ", taxi_own_veh["fuel_type"].max())
```

```
# Ex 2

wards_altered = pd.read_csv("Wards_Offices_Altered.csv")
wards_census_altered = wards_altered.merge(census, on = "ward")
print(wards_census_altered)

print("there are 46 rows")
```

```
#Ex 3
licenses = pd.read_pickle("licenses.p")
biz_owners = pd.read_pickle("business_owners.p")


licenses_owners = licenses.merge(biz_owners, on = "account")
print(licenses_owners.head())
print("-----------------------------------------------")

counted_df = licenses_owners.groupby("title").agg({"account" : "count"})
print(counted_df.head())
print("-----------------------------------------------")

sorted_df = counted_df.sort_values(["account"], ascending = False)
print(sorted_df.head())
```

```
grants_license = grants.merge(licenses, on = 'zip')
print(grants_licenses.loc[grants_licenses["business"]== "REGGIE'S BAR & GRILL",
                          ["grant", "company", "account", "ward", "business"]])
```

```
grants_license_ward = grants.merge(license, on = ["address", "zip"]) \
.merge(wards, on = "ward", suffixes = ("bus", "_wards"))
```

```
cal = pd.read_pickle("cta_calendar.p")
ridership = pd.read_pickle("cta_ridership.p")
stations = pd.read_pickle("stations.p")


ridership_cal_station = ridership.merge
(cal, on = [ "year", "month", "day"]).merge(stations, on =
                                          ["station_id"])

fil = ((ridership_cal_station [ "month" ] == 7)
      & (ridership_cal_station [ 'day_type'] == 'Weekday')
      & (ridership_cal_station [ "station_name"] == "Wilson"))

print(ridership_cal_station.loc[fil, 'rides'].sum())
```

```
140005
```

```
toy_story = pd.read_csv("toy_story.csv")
taglines = pd.read_pickle("taglines.p")

toy_story_tag = toy_story.merge(taglines, on = "id", how="left"

#toy_story_tag = toy_story.merge(taglines, on = "id")

toy_story_tag
```

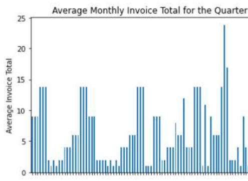
```
#Ex 8
import pandas as pd

# Load data
movies = pd.read_pickle('movies.p')

# Subset scifi_movies and action_movies
scifi_movies = movie_to_genres[movie_to_genres['genre'] == 'Science Fiction']
action_movies = movie_to_genres[movie_to_genres['genre'] == 'Action']

# Merge action_movies and scifi_movies with a right join and add suffixes
action_scifi = pd.merge(action_movies, scifi_movies, on='movie_id', how='right', suffixes=('_act', '_sci'))

# Subset rows where genre_act column is null (only science fiction)
scifi_only = action_scifi[action_scifi['genre_act'].isnull()]

# Merge movies and scifi_only with an inner join
result = pd.merge(movies, scifi_only, left_on='id', right_on='movie_id', how='inner')

# Display the result
print(result[['id', 'title']])
```

```
#Ex 9

import pandas as pd
import matplotlib.pyplot as plt

# Load data from CSV files
pop_movies = pd.read_csv('pop_movies.csv')
movie_to_genres = pd.read_csv('tdmb_movie_to_genres.csv')

# Merge using right join
genres_movies = pd.merge(movie_to_genres, pop_movies, left_on='movie_id', right_on='id', how='right')

# Group by genre and count the number of movies
genre_counts = genres_movies['genre'].value_counts().head(10)

# Plot the bar chart
plt.figure(figsize=(10, 6))
genre_counts.plot(kind='bar')
plt.xlabel('Genre')
plt.ylabel('Number of Movies')

plt.show()
```

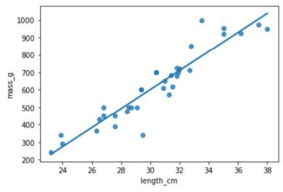
```
# Calculate total co2_emission per country: emissions_by_country
emissions_by_country = food.groupby('country')['co2_emission'].sum()
# Compute the first and third quantiles and IQR of emissions_by_country
q1 = np.quantile(emissions_by_country, 0.25)
q3 = np.quantile(emissions_by_country, 0.75)
iqr = q3 - q1
# Calculate the lower and upper cutoffs for outliers
lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr
# Subset emissions_by_country to find outliers
outliers = emissions_by_country[(emissions_by_country < lower) | (emissions_by_country > upper)]
print(outliers)
```

```
country
Argentina    2172.4
Name: co2_emission, dtype: float64
```

```
#Ex 12

import pandas as pd
import matplotlib.pyplot as plt

inv_jul = pd.read_csv("inv_jul.csv")
inv_aug = pd.read_csv("inv_aug.csv")
inv_sep = pd.read_csv("inv_sep.csv")

avg_inv_by_month = pd.concat([inv_jul, inv_aug, inv_sep], keys=['7Jul', '8Aug', '9Sep'])

average_totals = avg_inv_by_month.groupby('invoice_date')['total'].agg('mean')

average_totals.plot(kind='bar')
plt.xlabel('Month')
plt.ylabel('Average Invoice Total')
plt.title('Average Monthly Invoice Total for the Quarter')
plt.xticks(rotation=0)
plt.show()
```



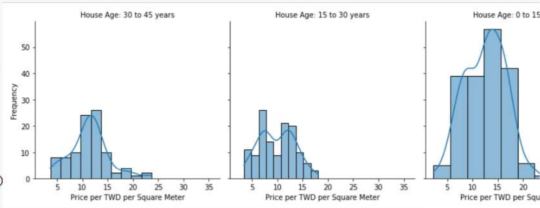
```
sns.regplot(x="length_cm",
            y ="mass_g",
            data=bream,
            ci=None)

plt.show()
```



```
#Ex 14

import pandas as pd
import matplotlib.pyplot as plt

unemployment = pd.read_csv("unemployment.csv")
inflation = pd.read_csv("inflation.csv")

inflation_unemploy = pd.merge_ordered(inflation, unemployment, on='date'
print(inflation_unemploy)

plt.scatter(inflation_unemploy['unemployment_rate'], inflation_unemploy['cpi'])
plt.xlabel('Unemployment Rate')
plt.ylabel('CPI (Inflation)')
plt.title('Scatter Plot of Unemployment Rate vs. CPI (Inflation)')
plt.show()
```

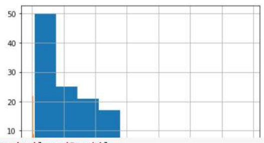
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

rice_consumption = food_consumption(food_consumption["food_category"] == "rice")

rice_consumption("co2_emission")
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have loaded the taiwan_real_estate DataFrame

# Create a histogram with 3 panels (one for each house_age_years group)
g = sns.FacetGrid(taiwan_real_estate, col='house_age_years', col_wrap=3, height=4)
g.map(sns.histplot, 'price_twd_msq', bins=10, kde=True)

# Set labels and titles
g.set_axis_labels("Price per TWD per Square Meter", "Frequency")
g.set_titles("House Age: {col_name} years")

# Show the plot
plt.show()
```



```
print(food.groupby("food_category") ['co2_emission'].agg([np.var, np.std]))

import matplotlib.pyplot as plt
print(food[food["food_category"] == 'beef'] ['co2_emission'].hist())
print(food[food["food_category"] == "eggs"] ['co2_emission'].hist())
```

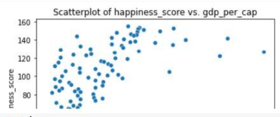
```
                       var         std
food_category
beef          88748.408132  297.906710
dairy         17671.891985  132.935669
eggs             21.371819    4.622966
fish            921.637349   30.358481
lamb_goat     16475.518363  128.356996
nuts             35.639652    5.969895
pork           3094.963537   55.632396
poultry         245.026801   15.653332
rice           2281.376243   47.763754
soybeans          0.879882    0.938020
wheat            71.023937    8.427570
AxesSubplot(0.125,0.125;0.775x0.755)
AxesSubplot(0.125,0.125;0.775x0.755)
```



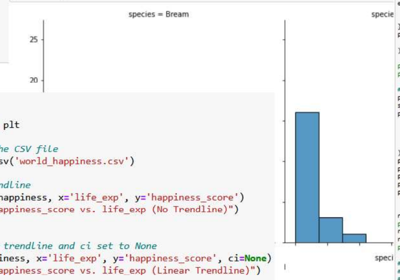
```
perch = fish[fish['species'] == 'Perch']
print(perch.head())
sns.regplot(x='length_cm', y = 'mass_g', data = perch, ci=None)
plt.show()
perch['length_cm_cubed'] = perch['length_cm']**3
sns.regplot(x='length_cm_cubed', y = 'mass_g', data = perch, ci=None)
plt.show()
mdl_perch = ols('mass_g ~ length_cm_cubed', data=perch).fit()
print(mdl_perch.params)
explanatory_data = pd.DataFrame({'length_cm_cubed': np.arange(10,41,5)**3,
'length_cm': np.arange(10,41,5)})
prediction_data = explanatory_data.assign(mass_g=mdl_perch.predict(explanatory_data))
print(prediction_data)
fig = plt.figure()
sns.regplot(x='length_cm_cubed', y = 'mass_g', data = perch, ci=None)
sns.scatterplot(data=prediction_data, x = 'length_cm_cubed',
                y = "mass_g", color='red', marker='s')

plt.show()
```

```
   species  mass_g  length_cm
55   Perch     5.9        7.5
56   Perch    32.0       12.5
57   Perch    40.0       13.8
58   Perch    51.5       15.0
59   Perch    70.0       15.7
```

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Load the DataFrame from the CSV file
world_happiness = pd.read_csv('world_happiness.csv')

# Scatterplot of happiness_score versus gdp_per_cap
sns.scatterplot(data=world_happiness, x='gdp_per_cap', y='happiness_score')
plt.title("Scatterplot of happiness_score vs. gdp_per_cap")
plt.show()

# Calculate the correlation between gdp_per_cap and happiness_score
cor_gdp = world_happiness['gdp_per_cap'].corr(world_happiness['happiness_score'])
print(f"Correlation between gdp_per_cap and happiness_score: {cor_gdp:.2f}")

# Add a new column 'log_gdp_per_cap' to world_happiness with the log transformation
world_happiness['log_gdp_per_cap'] = np.log(world_happiness['gdp_per_cap'])

# Seaborn scatterplot of happiness_score versus log_gdp_per_cap
sns.scatterplot(data=world_happiness, x='log_gdp_per_cap', y='happiness_score')
plt.title("Scatterplot of happiness_score vs. log_gdp_per_cap")
plt.show()

# Calculate the correlation between log_gdp_per_cap and happiness_score
cor_log_gdp = world_happiness['log_gdp_per_cap'].corr(world_happiness['happiness_score'])
print(f"Correlation between log_gdp_per_cap and happiness_score: {cor_log_gdp:.2f}")
```



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

fish = pd.read_csv("fish.csv")

sns.displot(data=fish, x='mass_g', col="species", col_wrap=2, bins=9)

plt.show()
```



```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Load the DataFrame from the CSV file
world_happiness = pd.read_csv('world_happiness.csv')

# Scatterplot without a trendline
sns.scatterplot(data=world_happiness, x='life_exp', y='happiness_score')
plt.title("Scatterplot of happiness_score vs. life_exp (No Trendline)")
plt.show()

# Scatterplot with a linear trendline and ci set to None
sns.regplot(data=world_happiness, x='life_exp', y='happiness_score', ci=None)
plt.title("Scatterplot of happiness_score vs. life_exp (Linear Trendline)")
plt.show()

# Calculate the correlation between life_exp and happiness_score
cor = world_happiness['life_exp'].corr(world_happiness['happiness_score'])

print(f"Correlation between life_exp and happiness_score: {cor:.2f}")
```

Scatterplot of happiness_score vs. life_exp (No Trendline)
160
140
120

```
summary_stats = fish.groupby("species") ['mass_g'].mean()
print(summary_stats)
```

```
species
Bream    617.828571
Perch    382.239286
Pike     718.705882
Roach    152.050000
Name: mass_g, dtype: float64
```

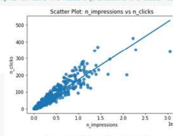
```
from statsmodels.formula.api import ols
mdl_mass_vs_species = ols('mass_g ~ species', data = fish).fit()

print(mdl_mass_vs_species.params)
```

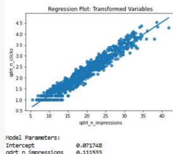
```
Intercept           617.828571
species[T.Perch]   -235.589286
species[T.Pike]     100.877311
species[T.Roach]   -465.778571
dtype: float64
```

```
mdl_mass_vs_species = ols(
"mass_g ~ species", data=fish).fit()

print(mdl_mass_vs_species.params)
```

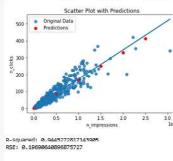
```
Intercept           617.828571
species[T.Perch]   -235.589286
species[T.Pike]     100.877311
species[T.Roach]   -465.778571
dtype: float64
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.formula.api import ols

ad_conversion = pd.read_csv("ad_conversion.csv")

sns.regplot(x="n_impressions", y="n_clicks", data=ad_conversion, ci=None)
plt.title("Scatter Plot: n_impressions vs n_clicks")
plt.xlabel("n_impressions")
plt.ylabel("n_clicks")
plt.show()

ad_conversion["qdrt_n_impressions"] = ad_conversion["n_impressions"] ** 0.25
ad_conversion["qdrt_n_clicks"] = ad_conversion["n_clicks"] ** 0.25
sns.regplot(x="qdrt_n_impressions", y="qdrt_n_clicks", data=ad_conversion, ci=None)
plt.title("Regression Plot: Transformed Variables")
plt.xlabel("qdrt_n_impressions")
plt.ylabel("qdrt_n_clicks")
plt.show()

# Do the points track the line more closely? Yes.

mdl_click_vs_impression = ols("qdrt_n_clicks ~ qdrt_n_impressions", data=ad_conversion).
print("Model Parameters:")
print(mdl_click_vs_impression.params)

explanatory_data = pd.DataFrame({
    "qdrt_n_impressions": np.arange(0, 3000000, 500000) ** 0.25,
    "n_impressions": np.arange(0, 3000000, 500000)
})
prediction_data = explanatory_data.assign(
    qdrt_n_clicks=mdl_click_vs_impression.predict(explanatory_data)
)

print("Prediction Data:")
print(prediction_data)

# Back transformation:
prediction_data["n_clicks"] = prediction_data["qdrt_n_clicks"] ** 4
sns.regplot(x="n_impressions", y="n_clicks", data=ad_conversion, ci=None, label="Origina
plt.scatter(
    x=prediction_data["n_impressions"],
    y=prediction_data["n_clicks"],
    color="red",
    label="Predictions",
)
plt.title("Scatter Plot with Predictions")
plt.xlabel("n_impressions")
plt.ylabel("n_clicks")
plt.legend()
plt.show()


r_squared = mdl_click_vs_impression.rsquared
print("R-squared:", r_squared)
residuals = ad_conversion["qdrt_n_clicks"] - mdl_click_vs_impression.fittedvalues
rse = np.sqrt(np.sum(residuals ** 2) / (len(ad_conversion) - 2))
print("RSE:", rse)

# Interpret the result:
# - A higher R-squared value indicates a better fit, with 1.0 being a perfect fit.
# - RSE measures the standard deviation of the residuals, indicating how well the model
# - A lower RSE suggests a better fit, as it means the model's predictions are closer to
```



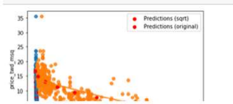


```
Model Parameters:
Intercept             0.071748
qdrt_n_impressions    0.111533
dtype: float64
Prediction Data:
   qdrt_n_impressions  n_impressions  qdrt_n_clicks
0            0.000000              0       0.071748
1           26.591479         500000       3.037576
2           31.622777        1000000       3.598732
3           34.996355        1500000       3.974998
4           37.606031        2000000       4.266063
5           39.763536        2500000       4.506696
```



```
R-squared: 0.9445272817143905
RSE: 0.19690640896875727
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.formula.api import ols

# Assuming you have already loaded your dataset as "taiwan_real_estate"

# Create sqrt_dist_to_mrt_m
taiwan_real_estate["sqrt_dist_to_mrt_m"] = np.sqrt(taiwan_real_estate["dist_to_mrt_m"])

# Create a linear regression model
mdl_price_vs_dist = ols("price_twd_msq ~ sqrt_dist_to_mrt_m", data=taiwan_real_estate).fit()

# Create explanatory_data
explanatory_data = pd.DataFrame({
    "sqrt_dist_to_mrt_m": np.sqrt(np.arange(0, 81, 10) ** 2),
    "dist_to_mrt_m": np.arange(0, 81, 10) ** 2
})

# Create prediction_data by adding a column of predictions to explanatory_data
prediction_data = explanatory_data.assign(
    price_twd_msq=mdl_price_vs_dist.predict(explanatory_data)
)

# Create the initial regplot with square root values
sns.regplot(x="sqrt_dist_to_mrt_m", y="price_twd_msq", data=taiwan_real_estate, ci=None)

# Add a scatter plot for prediction_data with square root values (colored red)
plt.scatter(
    x=prediction_data["sqrt_dist_to_mrt_m"],
    y=prediction_data["price_twd_msq"],
    color="red",
    label="Predictions (sqrt)",
)

# Create a regplot with the original values
sns.regplot(x="dist_to_mrt_m", y="price_twd_msq", data=taiwan_real_estate, ci=None)

# Add a scatter plot for prediction_data with original values (colored red)
plt.scatter(
    x=prediction_data["dist_to_mrt_m"],
    y=prediction_data["price_twd_msq"],
    color="red",
    label="Predictions (original)",
)

# Show the updated plot
plt.legend()
plt.show()
```

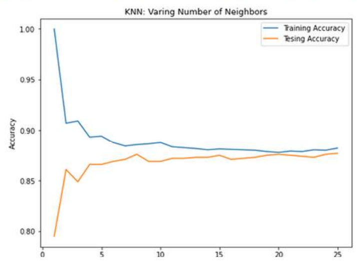


```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
                                                    random_state = 21,
                                                    stratify=y)

knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.861
```

```python
import matplotlib.pyplot as plt


train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1,26)
print(neighbors)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors = neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

#print(train_accuracies.values())
print(test_accuracies.values())
my_train = list(train_accuracies.values())
my_test = list(test_accuracies.values())

plt.figure(figsize=(8,6))
plt.title("KNN: Varing Number of Neighbors")
plt.plot(neighbors, my_train, label="Training Accuracy")
plt.plot(neighbors, my_test, label="Tesing Accuracy")
plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.show()
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 2
 25]
dict_values([0.795, 0.861, 0.849, 0.866, 0.866, 0.869, 0.871,
0.871, 0.872, 0.873, 0.875, 0.876, 0.875, 0.874, 0.873, 0.876,
```



```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

sales_df = pd.read_csv('sales_df.csv')

X = sales_df.drop(columns=['sales'])
y = sales_df['sales']

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Predicted values:", y_pred[:2])
print("Actual target :", y_test[:2])

r_squared = r2_score(y_test, y_pred)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print()

print("R-squared:", r_squared)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Predicted values: [53099.56399301 71056.14674591]
Actual target : [55261.28 67574.9 ]

R-squared: 0.9990147957135925
Root Mean Squared Error (RMSE): 2945.0531856107264
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression

sales_df = pd.read_csv('sales_df.csv')

X = sales_df[['radio', 'social_media']]
y = sales_df['sales']

kf = KFold(n_splits=6, shuffle=True, random_state=5)

reg = LinearRegression()
cv_scores = cross_val_score(reg, X, y, cv=kf)

print("CV scores: ", cv_scores)

mean_score = np.mean(cv_scores)
print("Mean:", mean_score)

std_score = np.std(cv_scores)
print("Std:", std_score)

confidence_interval = np.quantile(cv_scores, [0.025, 0.975]
print("95% Confidence Interval:", confidence_interval)
```

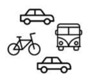
```
CV scores:  [0.74451678 0.77241887 0.76842114 0.7410406  0.
Mean: 0.7536937414361207
Std: 0.012305389070474664
95% Confidence Interval: [0.74141863 0.77191916]
```

What is statistics?

- The field of statistics – the practice and study of collecting and analyzing data.
- A summary statistics – a fact about or summary of some data.
- What can statistics do?
  - How likely is someone to purchase a product? Are people more likely to purchase it if they can use a different payment system?
  - How many occupants will your hotel have? How can you optimize occupancy?
  - How many sizes of jeans need to be manufactured so they can fit 95% of the population? Should the same number of each size be produced?
  - A/B tests: Which ad is more effective in getting people to purchase a product?

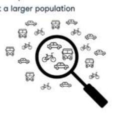
Types of statistics

Descriptive statistics
- *Describe* and summarize data



- 50% of friends drive to work
- 25% take the bus
- 25% bike

Inferential statistics
- Use a sample of data to make *inferences* about a larger population



What percent of people drive to work?

Being able to identify data types is important since the type of data you're working with will dictate what kinds of summary statistics and

Calculating variance

3. Sum squared distances

```python
sum_sq_dists = np.sum(sq_dists)
print(sum_sq_dists)
```

```
1624.065542
```

4. Divide by number of data points - 1

```python
variance = sum_sq_dists / (83 - 1)
print(variance)
```

```
19.805677
```

Use `np.var()`

```python
np.var(msleep['sleep_total'], ddof=1)
```

```
19.805677
```

*Without `ddof=1`, **population variance** is calculated instead of **sample variance**:*

```python
np.var(msleep['sleep_total'])
```

```
19.567055
```

```python
np.sqrt(np.var(msleep['sleep_total'], ddof=1))
```

```
4.450357
```

```python
np.std(msleep['sleep_total'], ddof=1)
```

```
4.450357
```

```python
np.quantile(msleep['sleep_total'], 0.5)
```

```
10.1
```

```python
# 75th percentile
seventy_fifth = salaries["Salary_USD"].quantile(0.75)

# 25th percentile
twenty_fifth = salaries["Salary_USD"].quantile(0.25)

# Interquartile range
salaries_iqr = seventy_fifth - twenty_fifth

print(salaries_iqr)
```

Measuring chance

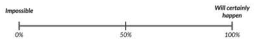
- We can measure the chances of an event using probability. We can calculate the probability of some event by taking the number of ways the event can happen and dividing it by the total number of possible outcomes.
- For example, if we flip a coin, it can land on either heads or tails. To get the probability of the coin landing on heads, we divide the 1 way to get heads by the two possible outcomes, heads and tails.

What's the probability of an event?

$$P(\text{event}) = \frac{\text{\# ways event can happen}}{\text{total \# of possible outcomes}}$$

Example: a coin flip

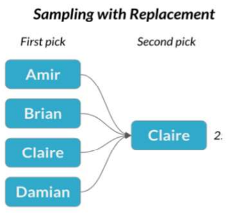
$$P(\text{heads}) = \frac{1 \text{ way to get heads}}{2 \text{ possible outcomes}} = \frac{1}{2} = 50\%$$



Independent events

*Two events are **independent** if the probability of the second event **isn't** affected by the outcome of the first event.*

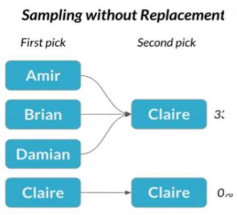
Sampling with replacement = each pick is independent



Dependent events

*Two events are **dependent** if the probability of the second event **is** affected by the outcome of the first event.*

Sampling without replacement = each pick is dependent



Central Limit Theorem in action

- The central limit theorem states that a sampling distribution of a sample statistic approaches the normal distribution as you take more samples, no matter the original distribution being sampled from.
- In this exercise, you'll focus on the sample mean and see the central limit theorem in action while examining the num_users column of amir_deals more closely, which contains the number of people who intend to use the product Amir is selling.

Types of data

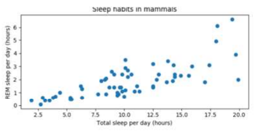
Numeric (Quantitative)
- **Continuous (Measured)**
  - Airplane speed
  - Time spent waiting in line
- **Discrete (Counted)**
  - Number of pets
  - Number of packages shipped

Categorical (Qualitative)
- **Nominal (Unordered)**
  - Married/unmarried
  - Country of residence
- **Ordinal (Ordered)**
  - ○ Strongly disagree
  - ○ Somewhat disagree
  - ○ Neither agree nor disagree
  - ● Somewhat agree
  - ○ Strongly agree

Categorical data can be represented as numbers

Nominal (Unordered)
- Married/unmarried (`1` / `0`)
- Country of residence (`1`, `2`, ...)

Ordinal (Ordered)
- Strongly disagree (`1`)
- Somewhat disagree (`2`)
- Neither agree nor disagree (`3`)
- Somewhat agree (`4`)
- Strongly agree (`5`)



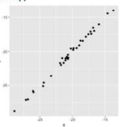
Correlation coefficient

- Quantifies the linear relationship between two variables
- Number between -1 and 1
- Magnitude corresponds to strength of relationship
- Sign (+ or -) corresponds to direction of relationship

- x = explanatory/independent variable
- y = response/dependent variable

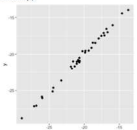
Magnitude = strength of relationship

0.99 (very strong relationship)

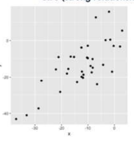


Magnitude = strength of relationship

0.99 (very strong relationship)

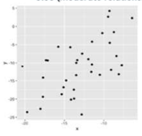


0.75 (strong relationship)



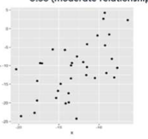
Magnitude = strength of relationship

0.56 (moderate relationship)

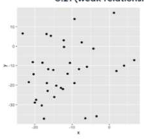


Magnitude = strength of relationship

0.56 (moderate relationship)

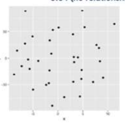


0.21 (weak relationship)



Magnitude = strength of relationship

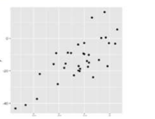
0.04 (no relationship)



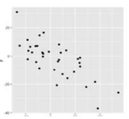
- Knowing the value of `x` doesn't tell us anything about `y`

Sign = direction

0.75: as `x` increases, `y` increases

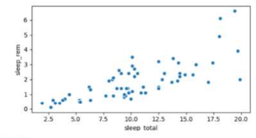


-0.75: as `x` increases, `y` decreases



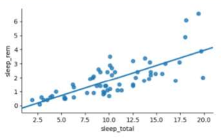
Visualizing relationships

```python
import seaborn as sns
sns.scatterplot(x="sleep_total", y="sleep_rem", data=msleep)
plt.show()
```

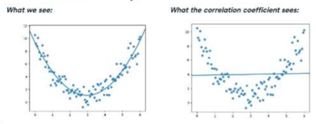


Adding a trendline

```python
import seaborn as sns
sns.lmplot(x="sleep_total", y="sleep_rem", data=msleep, ci=None)
plt.show()
```

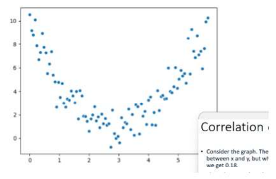


Non-linear relationships



Correlation caveats

- Consider the graph. There is clearly a relationship between x and y, but when we calculate the correlation, we get 0.18.
- This is because the relationship between the two variables is a quadratic relationship, not a linear relationship. The correlation coefficient measures the strength of linear relationships, and linear relationships only.



Log transformation

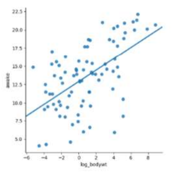
- When data is highly skewed like this, we can apply a log transformation.
- We'll create a new column called log_bodywt which holds the log of each body weight. We can do this using np.log. If we plot the log of bodyweight versus awake time, the relationship looks much more linear than the one between regular bodyweight and awake time. The correlation between the log of bodyweight and awake time is about 0.57, which is much higher than the 0.3 we had before.

Log transformation

```python
msleep['log_bodywt'] = np.log(msleep['bodywt'])

sns.lmplot(x='log_bodywt',
           y='awake',
           data=msleep,
           ci=None)
plt.show()
```

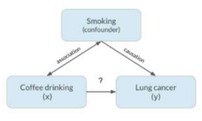
```python
msleep['log_bodywt'].corr(msleep['awake'])
```

```
0.5687943
```



A phenomenon called confounding can lead to spurious correlations. Let's say we want to know if drinking coffee causes lung cancer. Looking at the data, we find that coffee drinking and lung cancer are correlated, which may lead us to think that drinking more coffee will give you lung cancer.

Confounding



However, there is a third, hidden variable at play, which is smoking. It is also known that smoking causes lung cancer.

In reality, it turns out that coffee does not cause lung cancer and is only associated with it, but it appeared causal due to the third variable, smoking. This third variable is called a confounder, or lurking variable. This means that the relationship of interest between coffee and lung cancer is a spurious correlation.