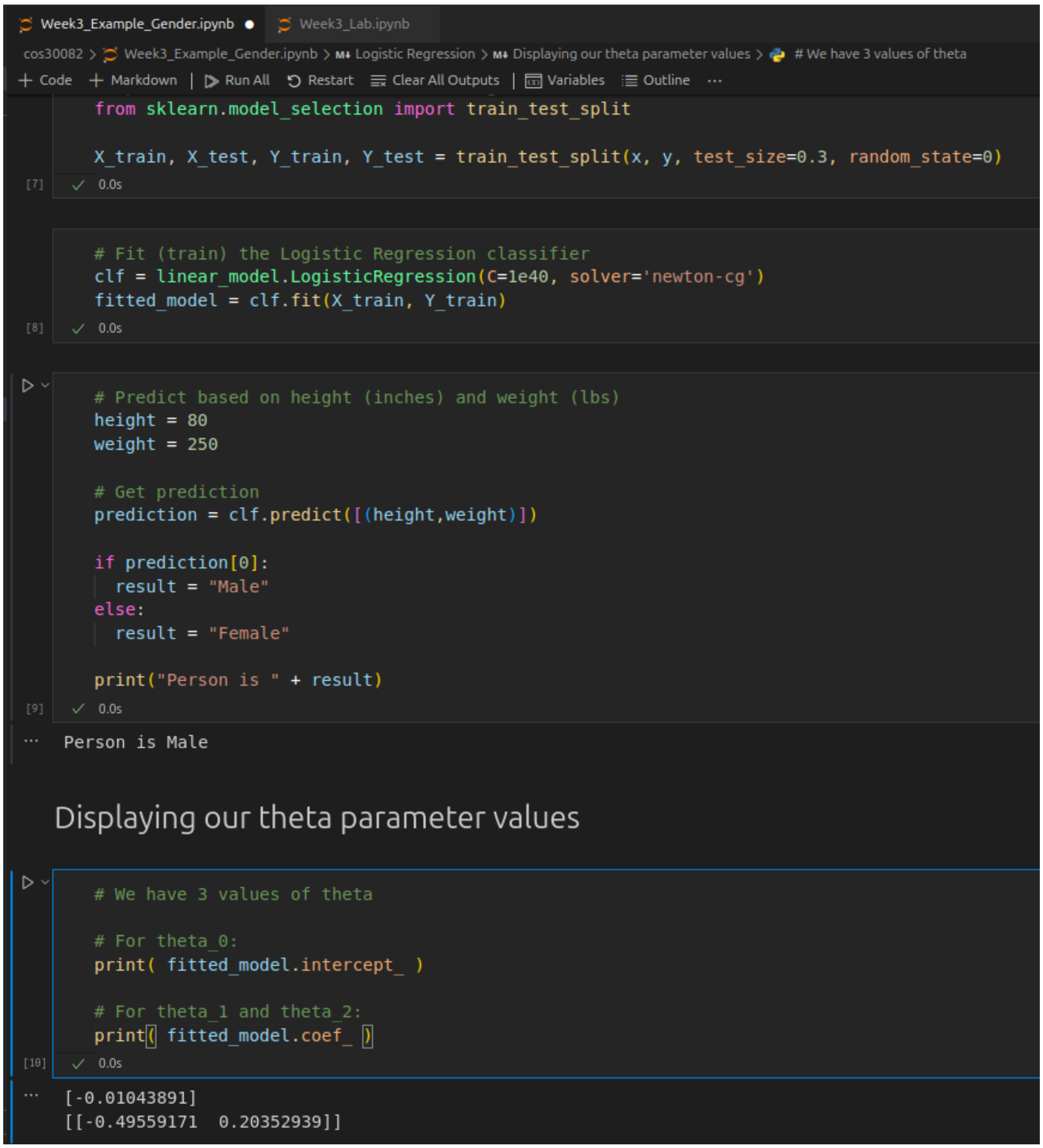
Week 3 Activities

Gender prediction model exploration

Execute all codes in the provided gender prediction on height-weight model



Titanic survivor prediction

Feature selection

The passenger id and name should not be an input attribute. Though the name contains one's title (Mr., Ms., ...) we have the 'sex' that bears almost the same meaning. Additionally, the 'ticket' column comprises different nominal data that is almost impossible to pre-process.

For the 'cabin' column, it is unclear how they are categorized (for instance, A and B for first class, ...). Therefore, without clear knowledge of this we might want to ignore this feature as well.

We use the drop() method with inplace parameter set to True from DataFrame class to remove unwanted attributes

```
df.head()
```

✓ 0.0s

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked_C	Embarked_Q	Emb
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	0	0	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	1	0	
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	0	0	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	0	0	
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	0	0	

```
# Remove unnecessary columns
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
df
```

✓ 0.0s

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S
0	0	3	1	22.0	1	0	7.2500	0	0	1
1	1	1	0	38.0	1	0	71.2833	1	0	0
2	1	3	0	26.0	0	0	7.9250	0	0	1
3	1	1	0	35.0	1	0	53.1000	0	0	1
4	0	3	1	35.0	0	0	8.0500	0	0	1
...	...	...	...	...	...	...	...	...	...	...
886	0	2	1	27.0	0	0	13.0000	0	0	1
887	1	1	0	19.0	0	0	30.0000	0	0	1
888	0	3	0	28.0	1	2	23.4500	0	0	1

Identify target variable

The target variable should whether the person survived or not, therefore is the 'survived' column

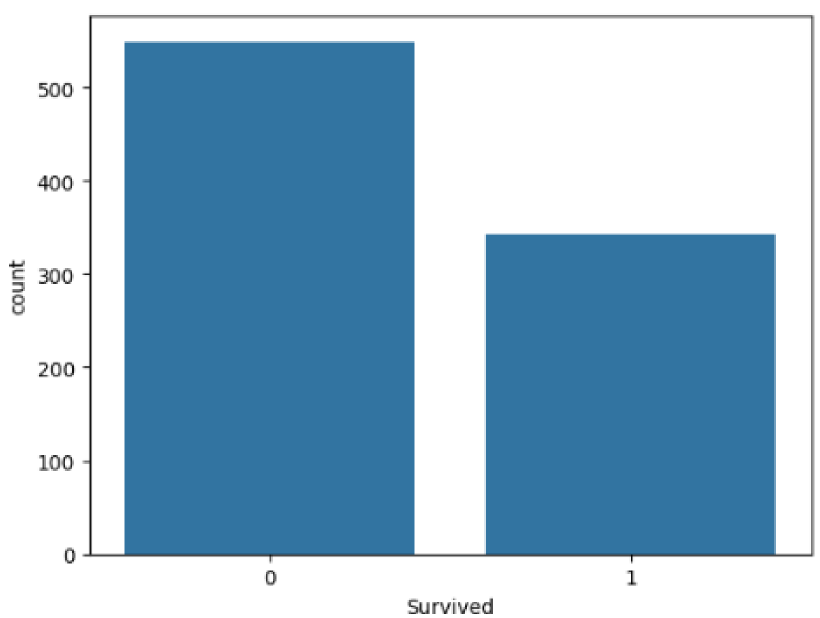
We split the data into input variables (x) and output variables (y) then split the data into training set and test set utilizing scikit-learn and pandas built-in methods.

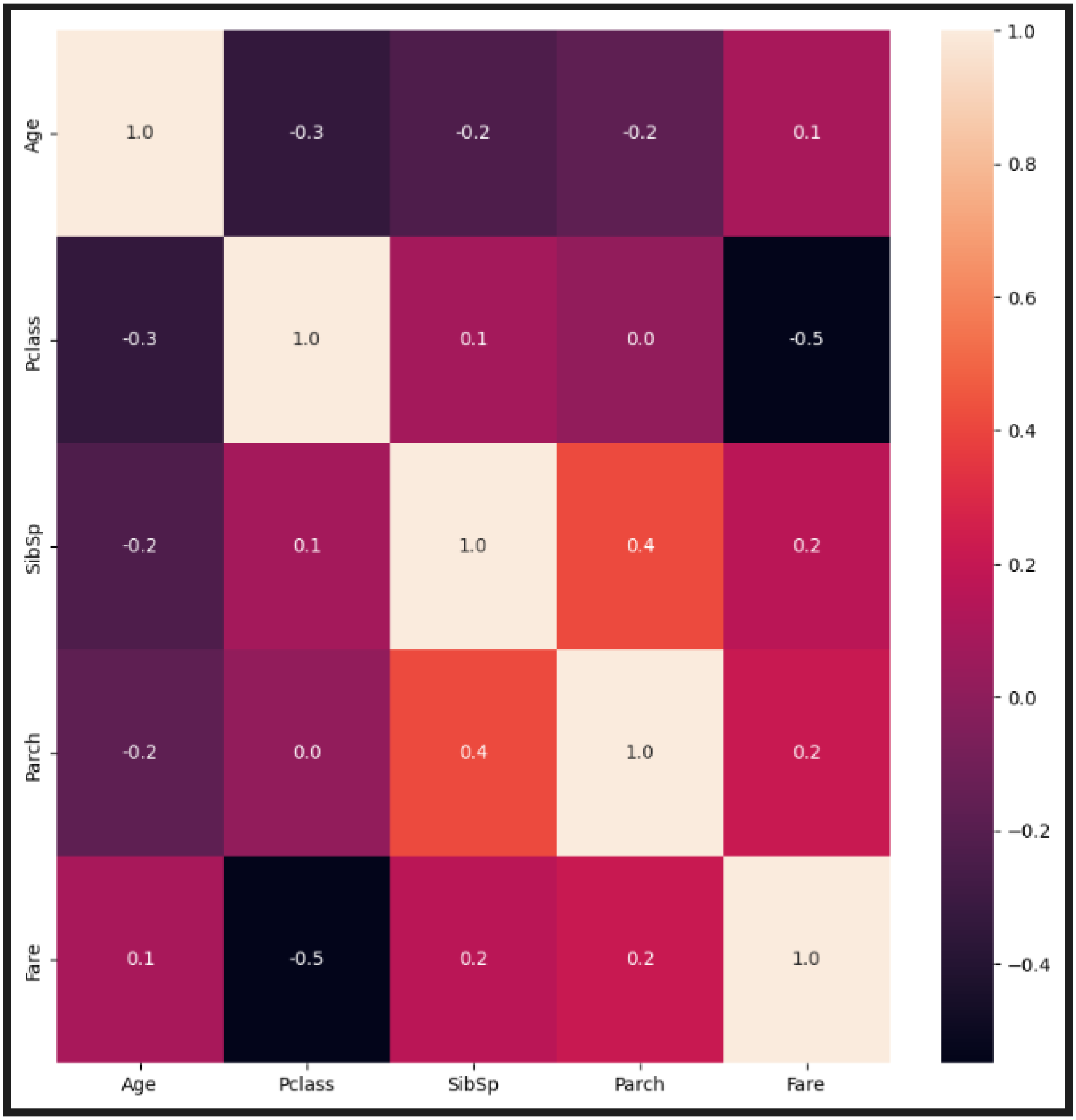
```python
x = df.drop('Survived', axis=1, inplace=False)
y = df['Survived'].copy()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
x_train
```
✓ 0.0s

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S
331	1	1	45.5	0	0	28.5000	0	0	1
733	2	1	23.0	0	0	13.0000	0	0	1
382	3	1	32.0	0	0	7.9250	0	0	1
704	3	1	26.0	1	0	7.8542	0	0	1
813	3	0	6.0	4	2	31.2750	0	0	1
...	...	...	...	...	...	...	...	...	...
106	3	0	21.0	0	0	7.6500	0	0	1
270	1	1	28.0	0	0	31.0000	0	0	1
860	3	1	41.0	2	0	14.1083	0	0	1
435	1	0	14.0	1	2	120.0000	0	0	1
102	1	1	21.0	0	1	77.2875	0	0	1

Data visualization and analysis

When building the model, it is important to preprocess and clean the data before fitting into the algorithms. Nevertheless, this is beyond the requirements and therefore will not be included in the report.





Logistic regression classifier model building and evaluation

Build train and observe the train result

```
# Create a Logistic Regression Classifier
from sklearn.linear_model import LogisticRegression
logistic_reg_clf = LogisticRegression()
logistic_reg_clf.fit(x_train, y_train)
print('Accuracy on test set:', logistic_reg_clf.score(x_test, y_test))
```

✓ 0.0s

```
Accuracy on test set: 0.8044692737430168
```

Predict on test set and observe evaluation metrics. The classification report in scikit-learn provide a combination of accuracy, precision, recall, support

```
from sklearn.metrics import classification_report
y_pred = logistic_reg_clf.predict(x_test)
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       105
           1       0.78      0.73      0.76        74

    accuracy                           0.80       179
   macro avg       0.80      0.79      0.80       179
weighted avg       0.80      0.80      0.80       179
```

Theta parameter values display

```
# Display the theta parameter values
print('Bias:', logistic_reg_clf.intercept_)
for col in x.columns:
    print(f'Theta {col}:', logistic_reg_clf.coef_[0][x.columns.get_loc(col)])
```

(variable) col: str

✓ 0.0s

```
Bias: [2.91254923]
Theta Pclass: -0.8136224070048483
Theta Sex: -2.496063117872217
Theta Age: -0.02522559632941104
Theta SibSp: -0.2449908951868847
Theta Parch: -0.12230174603848044
Theta Fare: 0.0032277460597245154
Theta Embarked_C: 1.2317089618931951
Theta Embarked_Q: 0.6828122297752853
Theta Embarked_S: 0.6415796983480684
```

Make prediction on random data

```python
import random

for i in range(5):
    record_idx = random.choice(x.index)
    record = x.loc[record_idx]
    pred = logistic_reg_clf.predict([record])[0]
    actual = y.loc[record_idx]
    print('Random Record:', record.to_dict())
    print('Prediction:', pred)
    print('Actual:', actual)
```

```
Random Record: {'Pclass': 3.0, 'Sex': 0.0, 'Age': 31.0, 'SibSp': 0.0, 'Parch': 0.0, 'Fare': 7.8542, 'Embarked_C': 0.0,
Prediction: 1
Actual: 0
Random Record: {'Pclass': 1.0, 'Sex': 1.0, 'Age': 28.0, 'SibSp': 0.0, 'Parch': 0.0, 'Fare': 47.1, 'Embarked_C': 0.0, 'E
Prediction: 0
Actual: 0
Random Record: {'Pclass': 2.0, 'Sex': 1.0, 'Age': 0.83, 'SibSp': 1.0, 'Parch': 1.0, 'Fare': 18.75, 'Embarked_C': 0.0, '
Prediction: 0
Actual: 1
Random Record: {'Pclass': 2.0, 'Sex': 1.0, 'Age': 52.0, 'SibSp': 0.0, 'Parch': 0.0, 'Fare': 13.5, 'Embarked_C': 0.0, 'E
Prediction: 0
Actual: 0
Random Record: {'Pclass': 2.0, 'Sex': 0.0, 'Age': 24.0, 'SibSp': 0.0, 'Parch': 0.0, 'Fare': 13.0, 'Embarked_C': 0.0, 'E
Prediction: 1
Actual: 1
```

Hyper parameter alternation

Alter train test split ratio

The modification in split ratio resulted in better accuracy when test set takes up 10%. Having said that, this ratio should never be the factor that makes the model to perform better or worse

```
# Alter split fraction to greater test size
x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(x, y, test_size=0.3, random_state=42)

logistic_reg_clf_2 = LogisticRegression()
logistic_reg_clf_2.fit(x_train_2, y_train_2)
y_pred_2 = logistic_reg_clf_2.predict(x_test_2)
print('Report on test set with 0.3 ratio:\n', classification_report(y_test_2, y_pred_2))

# Alter split fraction to smaller test size
x_train_3, x_test_3, y_train_3, y_test_3 = train_test_split(x, y, test_size=0.1, random_state=42)

logistic_reg_clf_3 = LogisticRegression()
logistic_reg_clf_3.fit(x_train_3, y_train_3)
y_pred_3 = logistic_reg_clf_3.predict(x_test_3)
print('Report on test set with 0.1 ratio:\n', classification_report(y_test_3, y_pred_3))
```

✓ 0.0s

```
Report on test set with 0.3 ratio:
              precision    recall  f1-score   support

           0       0.82      0.87      0.85       157
           1       0.80      0.73      0.76       111

    accuracy                           0.81       268
   macro avg       0.81      0.80      0.80       268
weighted avg       0.81      0.81      0.81       268

Report on test set with 0.1 ratio:
              precision    recall  f1-score   support

           0       0.88      0.85      0.87        54
           1       0.79      0.83      0.81        36

    accuracy                           0.84        90
   macro avg       0.84      0.84      0.84        90
weighted avg       0.85      0.84      0.85        90
```

Alter max iteration allowed

When the maximum iterations parameter in a logistic regression model is set to alarge value, the model tends to perform more efficiently because it allows the optimization process (like gradient descent) to converge fully. Yet this might suffer the risk of overfitting the data.

```python
# Increase maximum iteration
x_train_4, x_test_4, y_train_4, y_test_4 = train_test_split(x, y, test_size=0.2, random_state=42)

logistic_reg_clf_4 = LogisticRegression(max_iter=1000)
logistic_reg_clf_4.fit(x_train_4, y_train_4)
y_pred_4 = logistic_reg_clf_4.predict(x_test_4)
print('Report on test set with more epochs:\n', classification_report(y_test_4, y_pred_4))

# Decrease maximum iteration
x_train_5, x_test_5, y_train_5, y_test_5 = train_test_split(x, y, test_size=0.2, random_state=42)

logistic_reg_clf_5 = LogisticRegression(max_iter=10)
logistic_reg_clf_5.fit(x_train_5, y_train_5)
y_pred_5 = logistic_reg_clf_5.predict(x_test_5)
print('Report on test set with less epochs:\n', classification_report(y_test_5, y_pred_5))
```

✓ 0.1s

```
Report on test set with more epochs:
              precision    recall  f1-score   support

           0       0.83      0.86      0.84       105
           1       0.79      0.74      0.76        74

    accuracy                           0.81       179
   macro avg       0.81      0.80      0.80       179
weighted avg       0.81      0.81      0.81       179

Report on test set with less epochs:
              precision    recall  f1-score   support

           0       0.67      0.93      0.78       105
           1       0.79      0.35      0.49        74

    accuracy                           0.69       179
   macro avg       0.73      0.64      0.63       179
weighted avg       0.72      0.69      0.66       179
```

Alter both

Combine the most efficient parameters from above to create a model and observe the classification report.

```python
# Combine the optimal parameters
x_train_6, x_test_6, y_train_6, y_test_6 = train_test_split(x, y, test_size=0.1, random_state=42)
logistic_reg_clf_6 = LogisticRegression(max_iter=1000)
logistic_reg_clf_6.fit(x_train_6, y_train_6)
y_pred_6 = logistic_reg_clf_6.predict(x_test_6)
print('Report on test set with optimal parameters:\n', classification_report(y_test_6, y_pred_6))
```

✓ 0.1s

```
Report on test set with optimal parameters:
              precision    recall  f1-score   support

           0       0.88      0.85      0.87        54
           1       0.79      0.83      0.81        36

    accuracy                           0.84        90
   macro avg       0.84      0.84      0.84        90
weighted avg       0.85      0.84      0.85        90
```