In [26]:
```python
import numpy as np
import pandas as pd
import plotly
import plotly.figure_factory as ff
import plotly.graph_objs as go
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
```

In [27]:
```python
data = pd.read_csv(r'C:\Users\Akashraj D S\Downloads\8_LinearModels-20210710T061
data=data.iloc[:,1:]
```

In [28]:
```python
data.head()
```

Out[28]:

	f1	f2	f3	y
0	-195.871045	-14843.084171	5.532140	1.0
1	-1217.183964	-4068.124621	4.416082	1.0
2	9.138451	4413.412028	0.425317	0.0
3	363.824242	15474.760647	1.094119	0.0
4	-768.812047	-7963.932192	1.870536	0.0

In [29]:
```python
data.corr()['y']
```

Out[29]:
```
f1    0.067172
f2   -0.017944
f3    0.839060
y     1.000000
Name: y, dtype: float64
```

In [30]:
```python
data.corr()
```

Out[30]:

	f1	f2	f3	y
f1	1.000000	0.065468	0.123589	0.067172
f2	0.065468	1.000000	-0.055561	-0.017944
f3	0.123589	-0.055561	1.000000	0.839060
y	0.067172	-0.017944	0.839060	1.000000

In [31]:
```python
data.std()
```

Out[31]:
```
f1      488.195035
f2    10403.417325
f3        2.926662
y         0.501255
dtype: float64
```

In [32]:
```
X=data[['f1','f2','f3']].values
Y=data['y'].values
print(X.shape)
print(Y.shape)
```

```
(200, 3)
(200,)
```

What if our features are with different variance

```
* As part of this task you will observe how linear models work in case
 of data having feautres with different variance
* from the output of the above cells you can observe that var(F2)>>var
(F1)>>Var(F3)

> Task1:
    1. Apply Logistic regression(SGDClassifier with logloss) on 'data'
 and check the feature importance
    2. Apply SVM(SGDClassifier with hinge) on 'data' and check the feat
ure importance

> Task2:
    1. Apply Logistic regression(SGDClassifier with logloss) on 'data'
 after standardization
       i.e standardization(data, column wise): (column-mean(column))/st
d(column) and check the feature importance
    2. Apply SVM(SGDClassifier with hinge) on 'data' after standardizat
ion
       i.e standardization(data, column wise): (column-mean(column))/st
d(column) and check the feature importance
```

Make sure you write the observations for each task, why a particular feautre got more importance than others

Task 1

Applying Logistic Regression

In [33]:
```
from sklearn.linear_model import SGDClassifier
```

In [34]:
```
Log_reg_without_standardization = SGDClassifier(loss = 'log', random_state=42)
Log_reg_without_standardization.fit(X,Y)
```

Out[34]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=100
0,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=42, shuffle=True, tol=0.001, validation_fraction=0.
1,
              verbose=0, warm_start=False)
```

In [35]:
```
print(Log_reg_without_standardization.coef_)
```

```
[[ 8252.61712639 -9979.99939985 10367.64223133]]
```

Absolute value of weights tell us about the feature importance. If the value is more, the particular feature is more important than other

So, **f3 > f2 > f1**

Applying SVM

In [36]:
```
SVM_without_standardization = SGDClassifier(loss = 'hinge', random_state=42)
SVM_without_standardization.fit(X,Y)
```

Out[36]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=42, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [37]:
```
SVM_without_standardization.coef_
```

Out[37]:
```
array([[-7107.3738991 ,  9364.07983619,  9088.73593971]])
```

f2 > f3 > f1

Correlation between y and f3 is high

Correlation between y and f2, y and f1 is very low.

Since Correlation between y and f3 is high, f3 is important feature

Among f1 and f2, f2 is least important because of the high std it has

Weight values are high, outliers may be present, difficult to interpret

Task 2

Standardizing Dataset

In [38]:
```
features = list(data.columns)
features = features[0:len(features)-1]
features
```

Out[38]:
```
['f1', 'f2', 'f3']
```

In [39]:
```
data['f1']
```

Out[39]:
```
0      -195.871045
1     -1217.183964
2         9.138451
3       363.824242
4      -768.812047
          ...
195     119.423142
196     -37.805502
197     181.626647
198     443.199825
199     -51.189253
Name: f1, Length: 200, dtype: float64
```

In [40]:
```
for i in features:
    data[i] = (data[i] - data[i].mean())/data[i].std()
```

In [41]:
```
data
```

Out[41]:

	f1	f2	f3	y
0	-0.422067	-1.551708	0.181196	1.0
1	-2.514085	-0.515995	-0.200146	1.0
2	-0.002134	0.299269	-1.563735	0.0
3	0.724391	1.362511	-1.335214	0.0
4	-1.595658	-0.890469	-1.069923	0.0
...	...	...	...	...
195	0.223769	-0.411952	-1.391303	0.0
196	-0.098292	1.130524	0.143308	0.0
197	0.351185	0.180687	-0.663545	0.0
198	0.886981	-0.226199	0.159212	0.0
199	-0.125706	0.590425	1.546690	1.0

200 rows × 4 columns

In [42]:
```
data.corr()
```

Out[42]:

	f1	f2	f3	y
f1	1.000000	0.065468	0.123589	0.067172
f2	0.065468	1.000000	-0.055561	-0.017944
f3	0.123589	-0.055561	1.000000	0.839060
y	0.067172	-0.017944	0.839060	1.000000

In [43]:
```
X=data[['f1','f2','f3']].values
Y=data['y'].values
print(X.shape)
print(Y.shape)
```

```
(200, 3)
(200,)
```

Why Standardization and how it affects feature importance

Feature scaling is important when we use models that could use distance metrics for classification of the points.

Feature scaling brings all the values under Gaussian distribution.

Example: If the feature values are 3000 metres and 3 kms, eventhough they are same, 3000 gets higher value than 3, so it is necessary for feature-scaling

Applying Logistic Regression after Standardization

In [44]:
```
Log_reg_with_standardization = SGDClassifier(loss = 'log', random_state=42)
Log_reg_with_standardization.fit(X,Y)
```

Out[44]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=100
0,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=42, shuffle=True, tol=0.001, validation_fraction=0.
1,
              verbose=0, warm_start=False)
```

In [45]:
```
Log_reg_with_standardization.coef_
```

Out[45]:
```
array([[-2.8475733 ,  0.21386817,  8.8246235 ]])
```

Applying SVM after Standardization

In [46]:
```
SVM_with_standardization = SGDClassifier(loss = 'hinge',random_state=42)
SVM_with_standardization.fit(X,Y)
```

Out[46]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=42, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [47]:
```
SVM_with_standardization.coef_
```

Out[47]:
```
array([[-0.86745193,  1.39767463,  9.8957629 ]])
```

Coefficient values are less, easy to interpret

f3 is most important feature

In [ ]:
```

```