Take Home Exam Notebook

John A. Fonte, Esq.

In [1]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [42]:
```
# import data

data = pd.read_csv('D:/Github/Data-Science-Bootcamp/Previous Projects/Take Home Data Project/customers_data
df = data.copy() # I do this in case I screw up the original data in my analysis

df.head(4)
```

Out[42]:

	Unnamed: 0	purch_amt	gender	card_on_file	age	days_since_last_purch	loyalty
0	0	19.58	male	no	31.0	35.0	False
1	1	65.16	male	yes	23.0	61.0	False
2	2	40.60	female	no	36.0	49.0	False
3	3	38.01	male	yes	47.0	57.0	False

In [43]:
```
import seaborn as sns
```

Goal:

Using joining the loyalty program as the target variable, create a machine learning model that successfully predicts that.

In [44]:
```
# some brief preprocessing things:
df.drop(['Unnamed: 0'], axis=1, inplace=True)
```

In [45]:
```
df.columns = ['Purch_Amount', 'Gender', 'Card_on_File', 'Age', 'Days_Since_Last_Purch', 'Loyalty']
```

In [48]:
```
# binarizing column values
df['Gender'] = pd.Series([1 if i == 'male' else 0 for i in df.Gender])
df['Card_on_File'] = pd.Series([1 if i == 'yes' else 0 for i in df.Card_on_File])
df['Loyalty'] = pd.Series([1 if i == True else 0 for i in df.Loyalty])
```

In [49]:
```
df.head()
```

Out[49]:

	Purch_Amount	Gender	Card_on_File	Age	Days_Since_Last_Purch	Loyalty
0	19.58	1	0	31.0	35.0	0
1	65.16	1	1	23.0	61.0	0
2	40.60	0	0	36.0	49.0	0
3	38.01	1	1	47.0	57.0	0
4	22.32	0	1	5.0	39.0	0

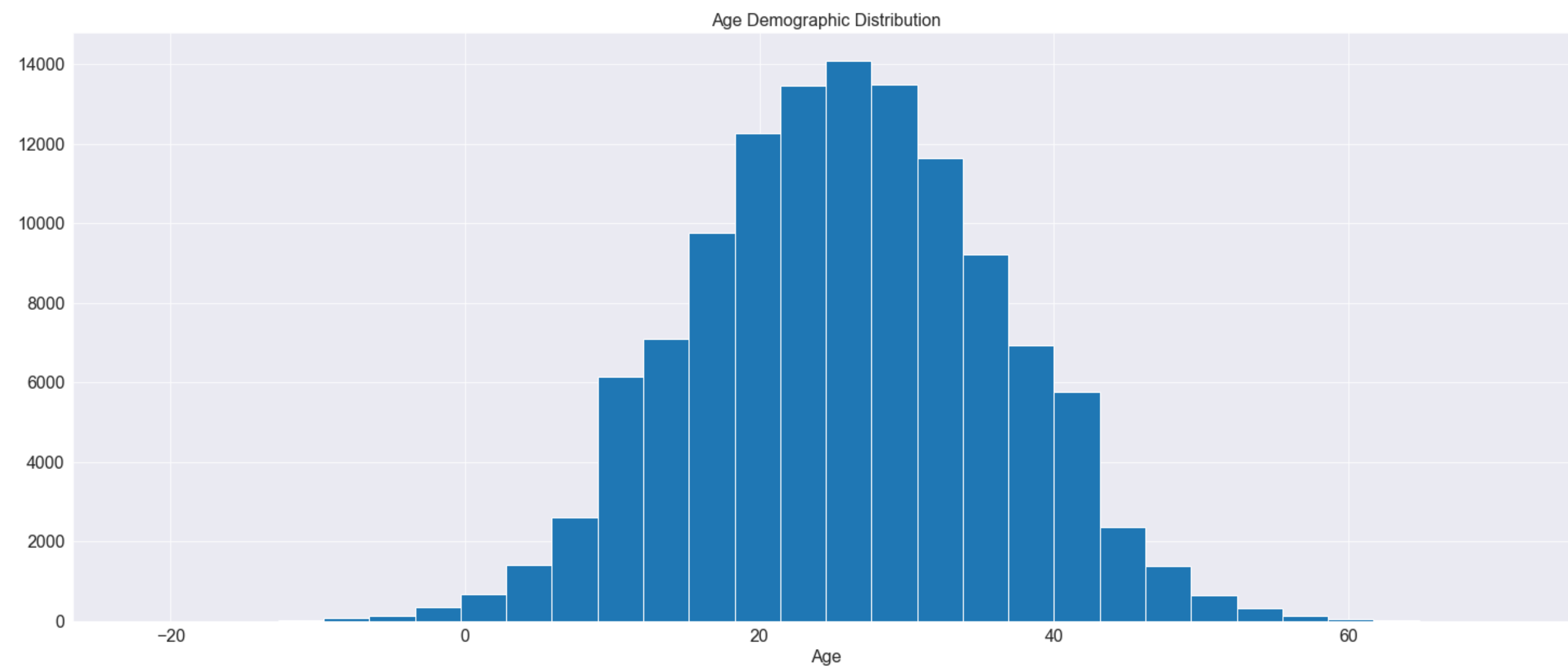
In [50]:
```
df.Loyalty.value_counts()
```

Out[50]:
```
0    100000
1     20000
Name: Loyalty, dtype: int64
```

In [13]:

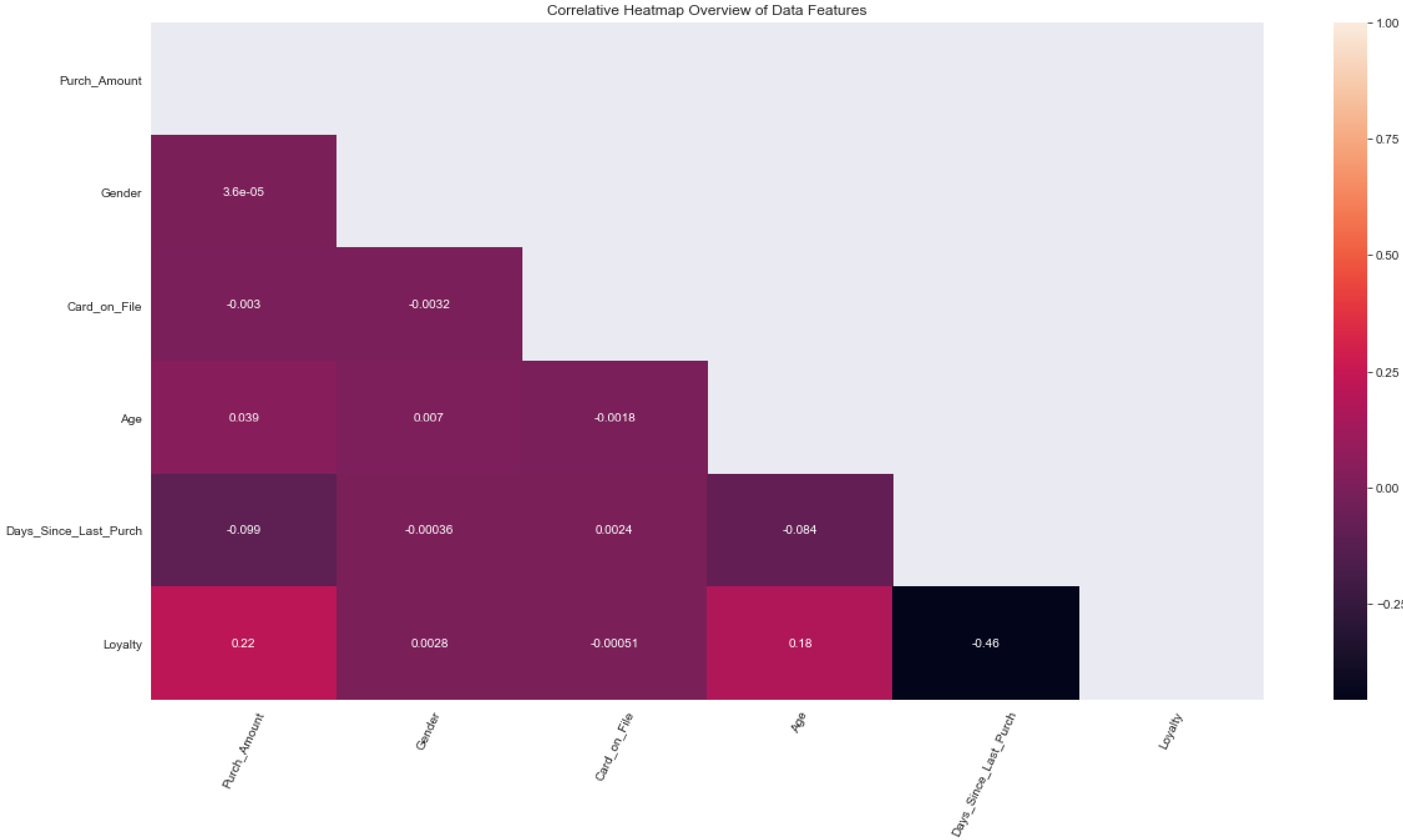
```
# Some data visualization
print(df.Loyalty.value_counts())
print(df.Gender.value_counts())

plt.figure(figsize=(25,10))
sns.set_style('darkgrid')
plt.hist(x=df.Age, bins=30)
plt.title('Age Demographic Distribution', fontsize=16)
plt.xlabel('Age', fontsize=16)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.show()
```

```
False    100000
True      20000
Name: loyalty, dtype: int64
male      60181
female    59819
Name: gender, dtype: int64
```

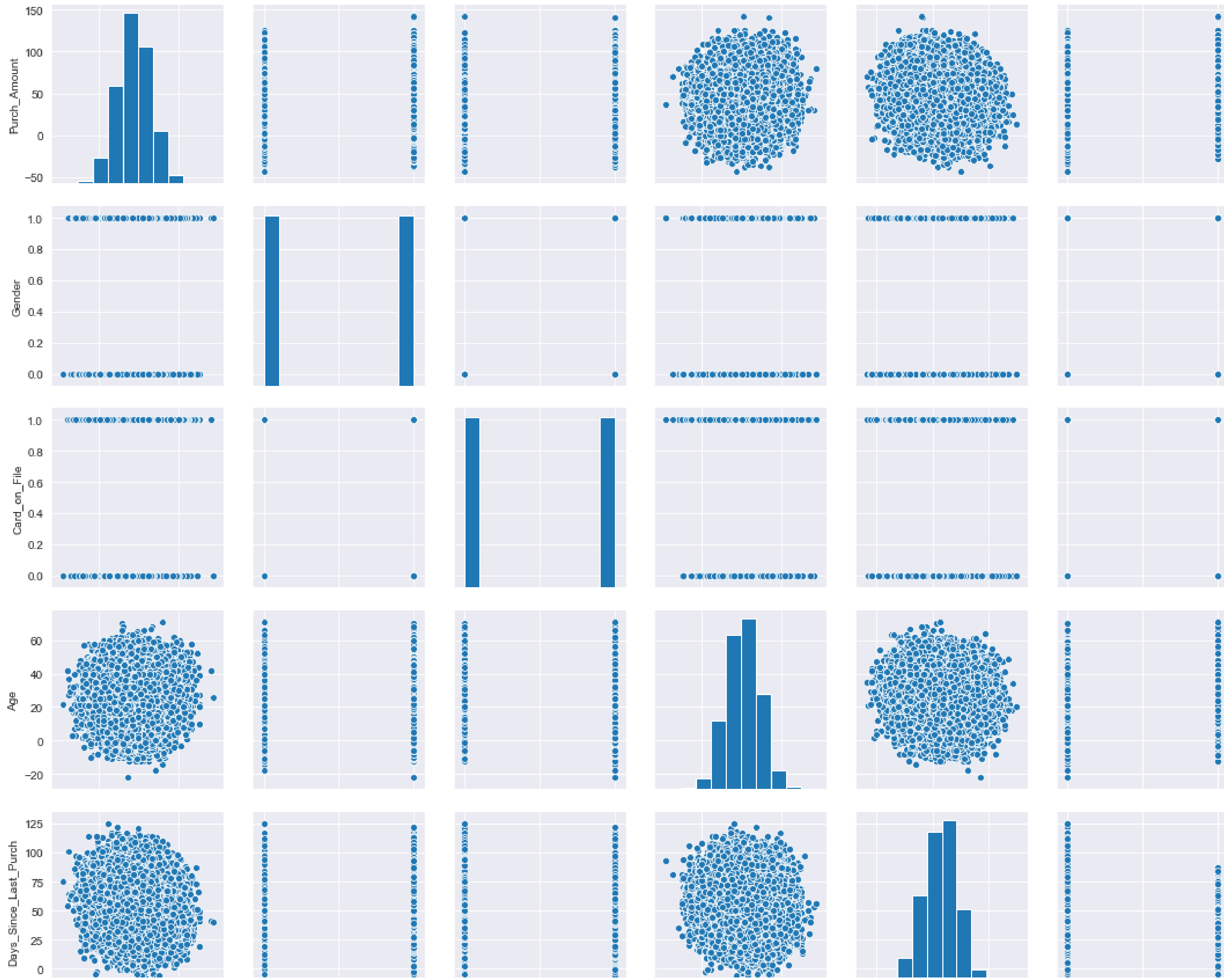


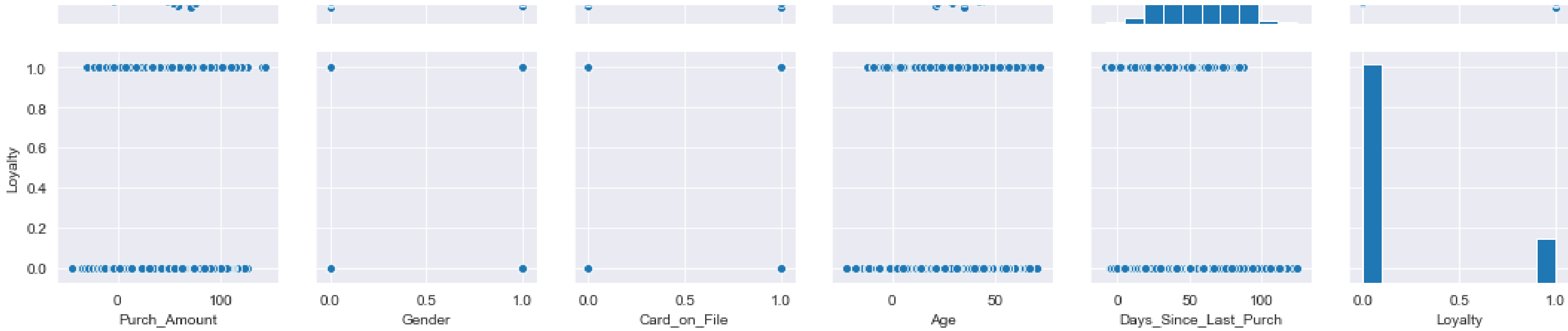
In [51]:

```
# checked the rest of the data --> it is even and normalized!

# checking correlations

correlations = df.corr()

mask = np.zeros_like(correlations, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

plt.figure(figsize=(20,10))
ax = sns.heatmap(correlations, mask=mask, xticklabels=df.columns,
                 yticklabels=df.columns, annot=True)

ax.set_xticklabels(labels=df.columns, rotation=65)

plt.title('Correlative Heatmap Overview of Data Features')

plt.show()
```


Correlative Heatmap Overview of Data Features
Purch_Amount
Gender
Card_on_File
Age
Days_Since_Last_Purch
Loyalty
3.6e-05
-0.003
-0.0032
0.039
0.007
-0.0018
-0.099
-0.00036
0.0024
-0.084
0.22
0.0028
-0.00051
0.18
-0.46
1.00
0.75
0.50
0.25
0.00
−0.25

In [52]:
```
sns.pairplot(data=df)
plt.show()
```





Data Cleaning

The three continuous variables - Purch_Amount, Age, and Days_Since_Last_Purch - all have negative values, which makes no sense. We can either choose to do two things: (1) deem the data useless and delete it to avoid skewing data, or (2) assume that the negative sign in front of the value was a typographical error and change it to a positive.

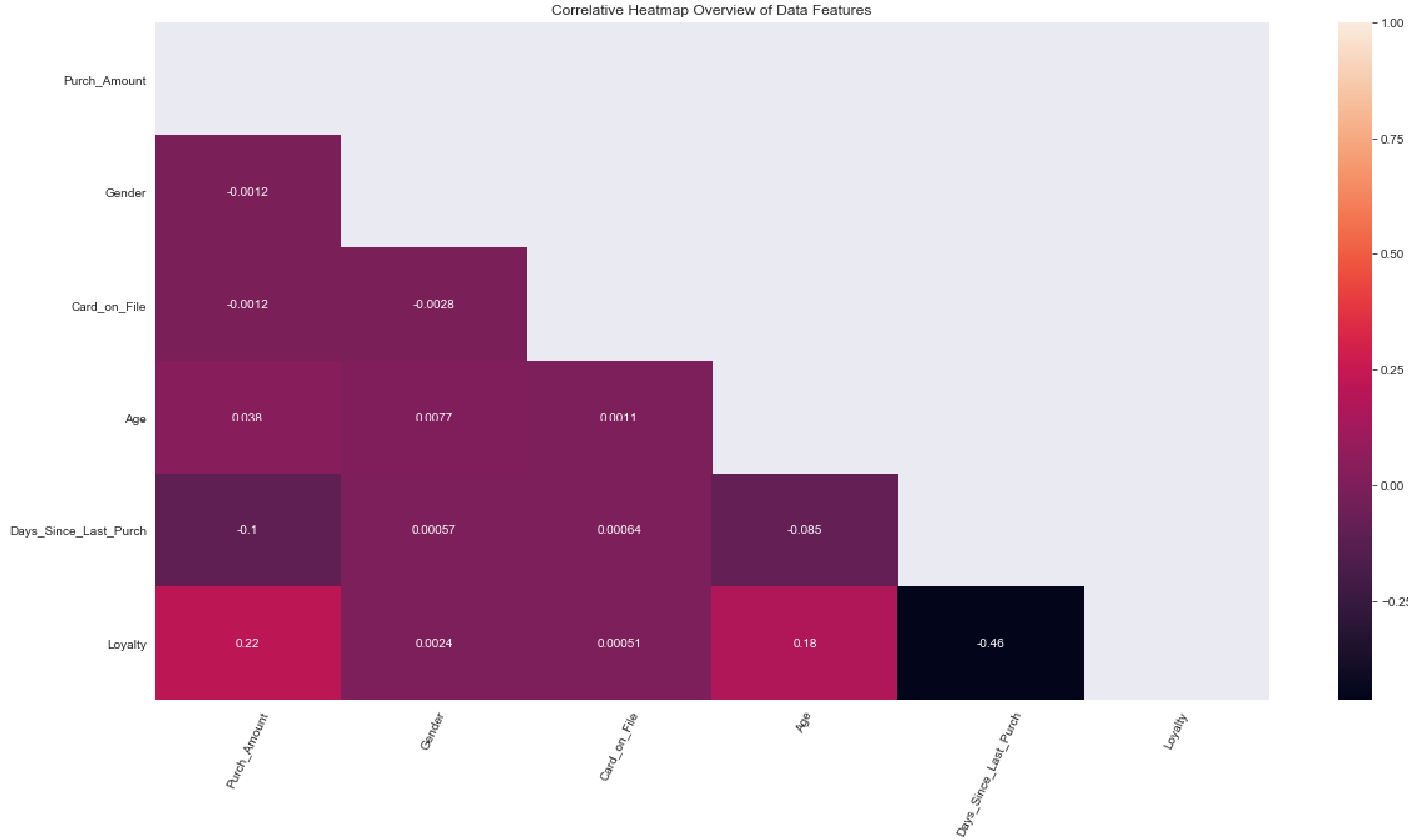
As much as I want to go with option (2), I have no basis for the assumption other than hopeful thinking. There is a significant amount of datapoints anyway, so I feel more comfortable with deleting those rows containing negative values in at least one of those columns.

In [56]:
```
# example of negative values
df.Age[df.Age < 10].value_counts()
```

Out[56]:
```
 9.0     1219
 8.0     1038
 7.0      873
 6.0      684
 5.0      562
 4.0      463
 3.0      376
 2.0      277
 1.0      229
 0.0      178
-1.0      145
-2.0      120
-3.0       92
-4.0       57
-5.0       40
-6.0       34
-7.0       26
-8.0       21
-9.0       16
-10.0      14
-12.0       6
-11.0       5
-18.0       1
-14.0       1
-22.0       1
Name: Age, dtype: int64
```

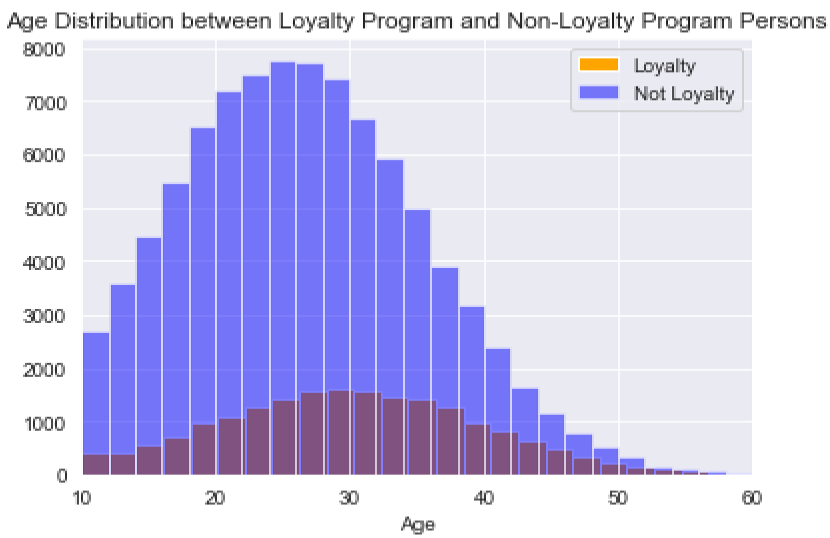
In [57]:
```
# systematically taking out the negative values:
df = df[df.Purch_Amount > 0]
df = df[df.Days_Since_Last_Purch > 0]
df = df[df.Age > 9] # I'm applying common sense, since customers are not going to be kids
```

In [58]:
```
# re-doing heatmap
# checked the rest of the data --> it is even and normalized!

# checking correlations

correlations = df.corr()

mask = np.zeros_like(correlations, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

plt.figure(figsize=(20,10))
ax = sns.heatmap(correlations, mask=mask, xticklabels=df.columns,
                 yticklabels=df.columns, annot=True)

ax.set_xticklabels(labels=df.columns, rotation=65)

plt.title('Correlative Heatmap Overview of Data Features')

plt.show()
```



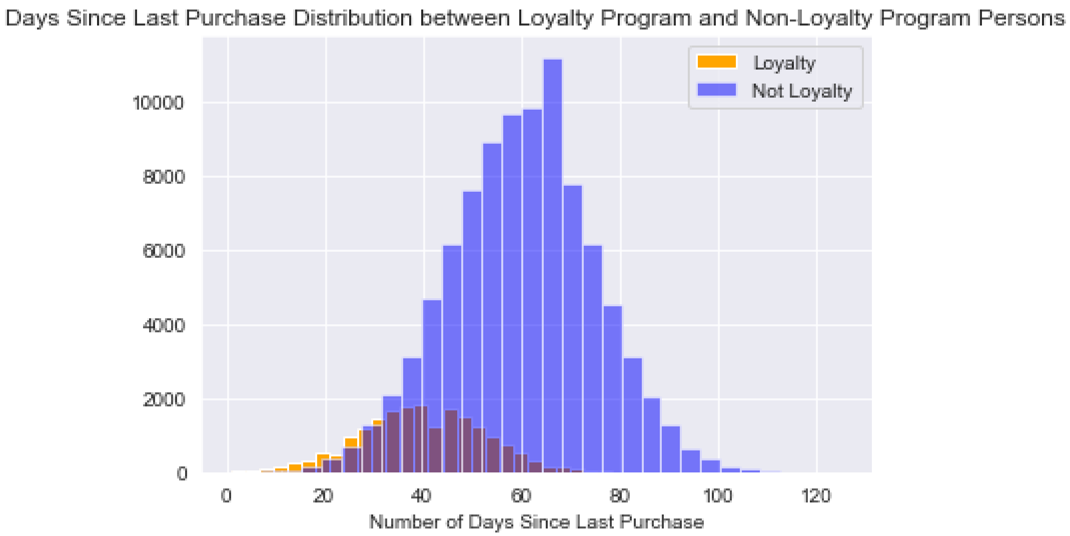
In [59]:
```
# redoing visualization
dfloyal = df[df.Loyalty == 1]
dfnotloyal = df[df.Loyalty == 0]
```

In [66]:
```
plt.hist(x=dfloyal.Age, color='orange', bins=30, alpha=1)
plt.hist(x=dfnotloyal.Age, color='blue', bins=30, alpha=0.5)
plt.legend(['Loyalty', 'Not Loyalty'])

plt.title('Age Distribution between Loyalty Program and Non-Loyalty Program Persons')
plt.xlabel('Age')
plt.xlim(10, 60)

plt.show()
```

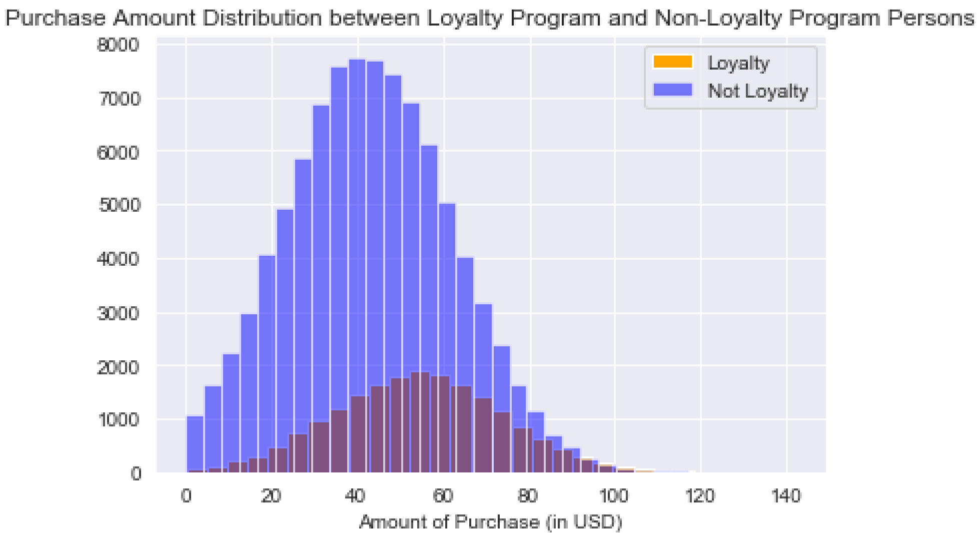


In [69]:

```
plt.hist(x=dfloyal.Days_Since_Last_Purch, color='orange', bins=30, alpha=1)
plt.hist(x=dfnotloyal.Days_Since_Last_Purch, color='blue', bins=30, alpha=0.5)
plt.legend(['Loyalty', 'Not Loyalty'])

plt.title('Days Since Last Purchase Distribution between Loyalty Program and Non-Loyalty Program Persons')
plt.xlabel('Number of Days Since Last Purchase')
#plt.xlim(10, 60)

plt.show()
```



In [70]:
```
plt.hist(x=dfloyal.Purch_Amount, color='orange', bins=30, alpha=1)
plt.hist(x=dfnotloyal.Purch_Amount, color='blue', bins=30, alpha=0.5)
plt.legend(['Loyalty', 'Not Loyalty'])

plt.title('Purchase Amount Distribution between Loyalty Program and Non-Loyalty Program Persons')
plt.xlabel('Amount of Purchase (in USD)')
#plt.xlim(10, 60)

plt.show()
```

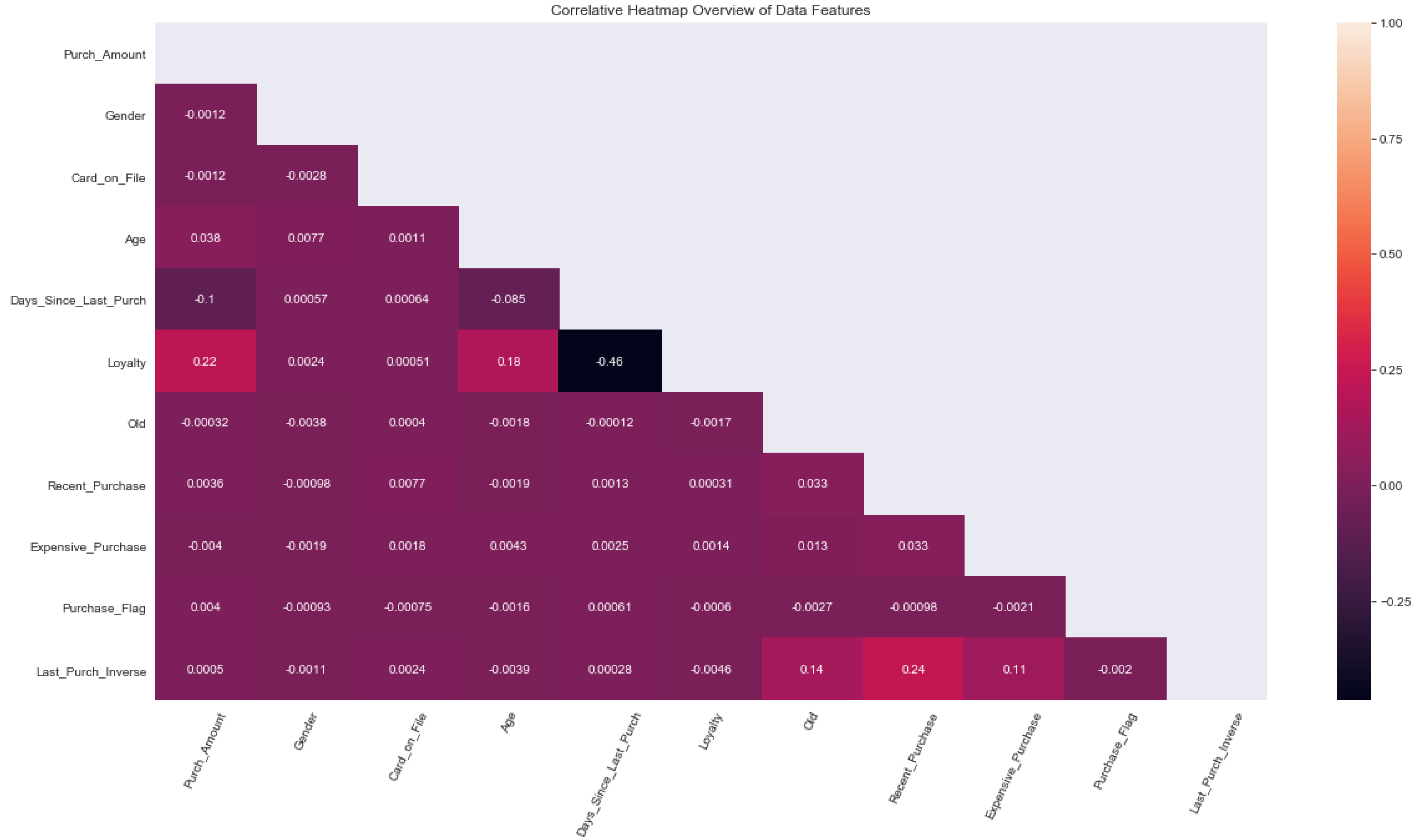


Feature Engineering

Based on the above distributions, I'm going to create binary features for certain thresholds, as seen below.

In [79]:
```
df['Old'] = pd.Series([1 if i > 25 else 0 for i in df.Age])
df['Recent_Purchase'] = pd.Series([1 if i < 25 else 0 for i in df.Days_Since_Last_Purch])
df['Expensive_Purchase'] = pd.Series([1 if i > 85 else 0 for i in df.Purch_Amount])

df['Purchase_Flag'] = pd.Series([1 if (i == 1 and j == 1) else 0 for i,j in zip(df.Recent_Purchase, df.Expe
```

In [93]:
```
# One final heatmap analysis

correlations = df.corr()

mask = np.zeros_like(correlations, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

plt.figure(figsize=(20,10))
ax = sns.heatmap(correlations, mask=mask, xticklabels=df.columns,
                 yticklabels=df.columns, annot=True)

ax.set_xticklabels(labels=df.columns, rotation=65)

plt.title('Correlative Heatmap Overview of Data Features')

plt.show()
```



Data Setup

In [94]:

```python
# setting up X and Y
X = df[['Purch_Amount', 'Age', 'Days_Since_Last_Purch']] # last one is strong negative correlation
Y = df['Loyalty']                                         # but strong correlation nonetheless
```

In [95]:
```
# splitting data

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, stratify=Y, random_state=666666)
```

In [125]:
```
# basic baseline test - bernoulli naive bayes

# instantiation
from sklearn.naive_bayes import BernoulliNB
bnb = BernoulliNB()

Y_pred = bnb.fit(X_train, Y_train).predict(X_test)

mislabeling = (Y_test != Y_pred).sum()

# Display our results.
print("Number of MISLABELED points out of a total {} points : {}".format(
    X_test.shape[0],
    mislabeling))
print('\nThat means a Naive Bayes accuracy of {}%.'.format((23074+4864 - mislabeling) / (23074+4864) * 100)

# Measure via confusion matrix

from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, Y_pred)

# confusion matrix layout:
# [[(True Positive), (False Negative)],
#  [(False Positive), (True Negative)]]
```

```
Number of MISLABELED points out of a total 27938 points : 4864

That means a Naive Bayes accuracy of 82.59002076025484%.
```

Out[125]:
```
array([[23074,     0],
       [ 4864,     0]], dtype=int64)
```

In [110]:
```
print('The null accuracy for this dataset is: {}%'.format(round(92295/(92295+19456)*100), 3))
```

```
The null accuracy for this dataset is: 83%
```

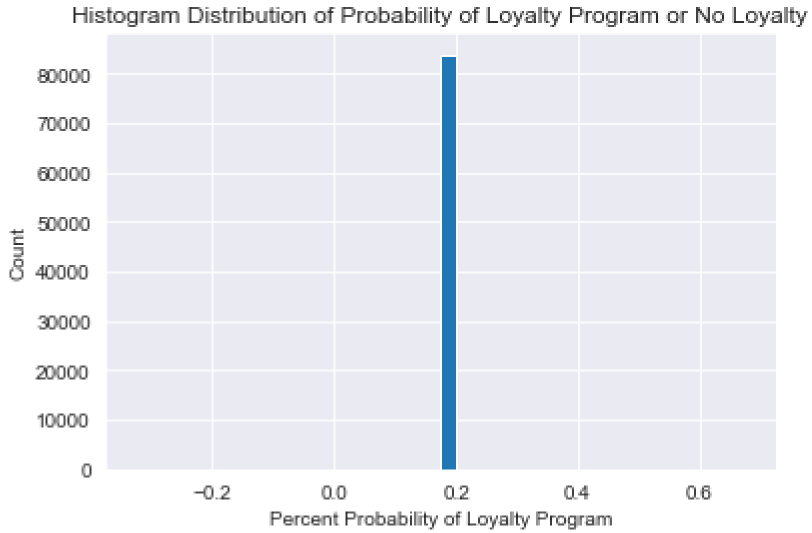
Analysis: Naive Bayes did not do better than the null accuracy. Let's see if more sophisticated machine learning models fare better.

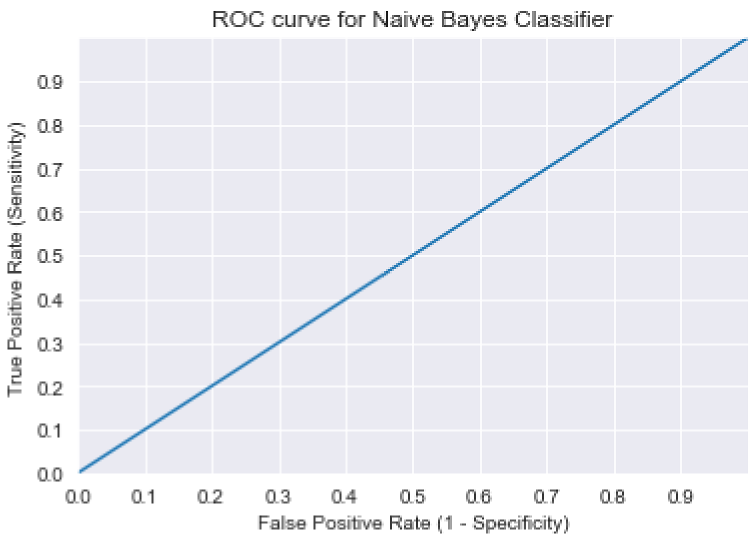
Binarization

Going to adjust binarization point based on ROC.

In [103]:

```python
from sklearn import metrics

# Determine probabilities

y_pred_prob = bnb.predict_proba(X_train)[:, 1] # using training data for better fitting!!!
y_pred_prob = y_pred_prob.reshape(-1,1)        # gotta reshape for it to work
# ^^^This means to predict probabilities of outcome from "class 1" for all rows

#good idea to graph out the probability distribution to get an idea of how to adjust
plt.hist(y_pred_prob, bins=40)
plt.title('Histogram Distribution of Probability of Loyalty Program or No Loyalty')
plt.ylabel('Count')
plt.xlabel('Percent Probability of Loyalty Program')
plt.show()

# How to determine this delineating point? How sensitive to make?
# Answer: ROC Curve.
# ROC Curve tells you what sensitivity/specificity tradeoff you will be able to get

fpr, tpr, thresholds = metrics.roc_curve(Y_train, y_pred_prob)
plt.plot(fpr, tpr)
plt.xlim(0,1) # from 0% to 100%
plt.ylim(0,1)
plt.title('ROC curve for Naive Bayes Classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.xticks(np.arange(0, 1, step=0.1))
plt.yticks(np.arange(0, 1, step=0.1))
plt.grid(True)


```





ummm...okay...

Supervised Modeling

- Logistic Regression
- KNN Classifier

- SVM Classifier
- AdaBooster

In [104]:
```
from sklearn.model_selection import cross_val_score
```

In [106]:
```
import warnings
warnings.filterwarnings(action='ignore')
```

In [107]:

```
# logistic regression
from sklearn.linear_model import LogisticRegression

# gridsearch would be great for this, but I found this faster for purposes of this project

logistregr0 = LogisticRegression(penalty='l2', C=0.05) # using L2, not L1 because of small feature space
logistregr1 = LogisticRegression(penalty='l2', C=0.25) # and large correlation with one of the features
logistregr2 = LogisticRegression(penalty='l2', C=0.65)
logistregr3 = LogisticRegression(penalty='l2', C=1)
logistregr4 = LogisticRegression(penalty='l2', C=1.5)

#----------------------

print('Logistic Regression scores with L2 C paramater = 0.05: ', cross_val_score(logistregr0, X, Y, cv=5))
print('\nLogistic Regression scores with L2 C paramater = 0.25: ', cross_val_score(logistregr1, X, Y, cv=5)
print('\nLogistic Regression scores with L2 C paramater = 0.65: ', cross_val_score(logistregr2, X, Y, cv=5)
print('\nLogistic Regression scores with L2 C paramater = 1: ', cross_val_score(logistregr3, X, Y, cv=5))
print('\nLogistic Regression scores with L2 C paramater = 1.5: ', cross_val_score(logistregr4, X, Y, cv=5))
```

```
Logistic Regression scores with L2 C paramater = 0.05:  [0.86309337 0.86263982 0.86246085 0.86563758 0.8656375
8]

Logistic Regression scores with L2 C paramater = 0.25:  [0.86309337 0.86246085 0.86228188 0.86572707 0.8656375
8]

Logistic Regression scores with L2 C paramater = 0.65:  [0.86300389 0.86250559 0.86219239 0.86568233 0.8656375
8]

Logistic Regression scores with L2 C paramater = 1:  [0.86304863 0.86250559 0.86214765 0.86568233 0.86563758]

Logistic Regression scores with L2 C paramater = 1.5:  [0.86304863 0.86250559 0.86214765 0.86572707 0.8656823
3]
```

Analysis: With a null accuracy of 83%, this is not that striking. Better than Naive Bayes though...

In [117]:

```
#KNN Classifier
from sklearn.neighbors import KNeighborsClassifier

knn1 = KNeighborsClassifier(n_neighbors=1)
knn2 = KNeighborsClassifier(n_neighbors=5)
knn3 = KNeighborsClassifier(n_neighbors=10)
knn4 = KNeighborsClassifier(n_neighbors=50)
knn5 = KNeighborsClassifier(n_neighbors=100)
knn6 = KNeighborsClassifier(n_neighbors=500)

#-------------------------------------------------------------------

print('Score for {}-Nearest Neighbors analysis:  \n{}'.format('1',
                                                              cross_val_score(knn1, X, Y, cv=5)))

print('\nScore for {}-Nearest Neighbors analysis:  \n{}'.format('5',
                                                                cross_val_score(knn2, X, Y, cv=5)))

print('\nScore for {}-Nearest Neighbors analysis:  \n{}'.format('10',
                                                                cross_val_score(knn3, X, Y, cv=5)))

print('\nScore for {}-Nearest Neighbors analysis:  \n{}'.format('50',
                                                                cross_val_score(knn4, X, Y, cv=5)))

print('\nScore for {}-Nearest Neighbors analysis:  \n{}'.format('100',
                                                                cross_val_score(knn5, X, Y, cv=5)))

print('\nScore for {}-Nearest Neighbors analysis:  \n{}'.format('500',
                                                                cross_val_score(knn6, X, Y, cv=5)))
```

```
Score for 1-Nearest Neighbors analysis:
[0.80340924 0.80389262 0.80791946 0.80420582 0.80268456]

Score for 5-Nearest Neighbors analysis:
[0.84434701 0.84192394 0.84635347 0.84514541 0.84675615]

Score for 10-Nearest Neighbors analysis:
[0.85562167 0.8541387  0.85404922 0.85530201 0.85673378]

Score for 50-Nearest Neighbors analysis:
[0.86255649 0.86165548 0.86040268 0.86375839 0.86259508]
```

```
Score for 100-Nearest Neighbors analysis:
[0.86242226 0.86210291 0.86214765 0.86532438 0.86375839]

Score for 500-Nearest Neighbors analysis:
[0.86291441 0.86362416 0.86228188 0.86505593 0.8652349 ]
```

Analysis: There is a diminishing return with accuracy when increasing neighbors. 87% is the highest that can probably be achieved.

In [119]:

```python
# SVM Classifier
from sklearn.svm import SVC

# using default 'rbf' kernel
# using decision function shape (one-vs.-rest, as opposed to one-vs.-one)
# using default tolerance for stopping criterion (0.001)

svm1 = SVC(C=0.01, class_weight='balanced', max_iter=2500)
svm2 = SVC(C=0.1, class_weight='balanced', max_iter=2500)
svm3 = SVC(C=0.5, class_weight='balanced', max_iter=2500)
svm4 = SVC(C=2, class_weight='balanced', max_iter=2500)

#-----------------------------------------------------

print('SVM scores with C-parameter of 0.01: \n', cross_val_score(svm1, X, Y, cv=3))
print('\nSVM scores with C-parameter of 0.1: \n', cross_val_score(svm2, X, Y, cv=3))
print('\nSVM scores with C-parameter of 0.5: \n', cross_val_score(svm3, X, Y, cv=3))
print('\nSVM scores with C-parameter of 2: \n', cross_val_score(svm4, X, Y, cv=3))

```

```
SVM scores with C-parameter of 0.01:
 [0.17411613 0.17409396 0.17409396]

SVM scores with C-parameter of 0.1:
 [0.17411613 0.17409396 0.17409396]

SVM scores with C-parameter of 0.5:
 [0.17414298 0.17409396 0.17422819]

SVM scores with C-parameter of 2:
 [0.17680062 0.17734228 0.18373154]
```

Analysis: Dismal. 2500 iterations is not enough time to converge. At the same time, when there is no limit to max_iter, it takes forever, to the point that it feels like the kernel is hanging.

In [120]:

```
from sklearn.ensemble import AdaBoostClassifier

# default n_estimators=50, so I'm really going for broke here!

adaboost1 = AdaBoostClassifier(n_estimators=250, learning_rate=0.01)
adaboost1 = AdaBoostClassifier(n_estimators=250, learning_rate=0.1)
adaboost3 = AdaBoostClassifier(n_estimators=250, learning_rate=1)

#-------------------------------------------------------------------

print('AdaBoost Classifier score with learning rate of {}: \n{}'.format('0.01',
                                                                         cross_val_score(adaboost1, X, Y, cv

print('\nAdaBoost Classifier score with learning rate of {}: \n{}'.format('0.1',
                                                                           cross_val_score(adaboost1, X, Y,

print('\nAdaBoost Classifier score with learning rate of {}: \n{}'.format('1',
                                                                           cross_val_score(adaboost3, X, Y,

```

```
AdaBoost Classifier score with learning rate of 0.01:
[0.86279563 0.86225503 0.86510067]

AdaBoost Classifier score with learning rate of 0.1:
[0.86279563 0.86225503 0.86510067]

AdaBoost Classifier score with learning rate of 1:
[0.86228558 0.86284564 0.86475168]
```

Analysis: No difference between learning rates. Still a frontrunner though for accuracies.

CONCLUSION

AdaBooster and high-K KNN classifiers are the best for modeling here.