Profiling for Authentication and Authorization

Course: CS795: Topics in Data Mining and Security

Team: Kiran Teja Sarvamthota (UIN: 01095632)

Hari Chandana Chintha (UIN: 01077086)

TABLE OF CONTENTS

INTRODUCTION

The basic objective of the project is to apply data mining techniques in a class project and to illustrate how data mining techniques implemented over a specified sequence of steps to achieve a stratified solution for the given problem statement.

The problem statement is analyzed thoroughly, and the data analysis is done so as to employ the data mining techniques and bring out a possible solution.

PROBLEM STATEMENT:

The input data set is provided with the User related data. It typically consists of historical login and access data for all 20 users in a department. The main objective is to develop a profile for each user.

The profile of user would specify different parameters of the User such as login, logoff, session time patterns as well as access patterns which as information about the user programs, library programs, files and printers. Relevant statistics along with association rules from the data are used to develop a profile for the user.

APPROACH:

The approach of this project is detailed in the below mentioned information.
Programming Language: Python
Environment: Jupyter notebook
Tool: Weka

The goal of building user profile from the given data is achieved in the following steps:

1. Raw data collection
2. Data preprocessing
3. Data cleaning
4. Exploratory data analysis
5. Build data model

The methodology is explained in detail in the below mentioned sequence of steps to achieve building the data product which is user profile in this case.

1. **Raw data collection**: The given input data set “proj-data.xlsx” is collected and analyzed. Looking into the specified data and understanding the major findings in the data set. The input data excel file is loaded into the python environment and set into pandas data frames.

COLUMN	FORMAT
User ID	U01-U20
User program ID	UP001-UP500
Library program/utility ID	LP001-LP100
File ID	F0001-F2000
Printer ID	PR1-PR6
E-mail program ID	E1-E5
Host machines	M01-M30
Date	MMDDYY
Time	HHMMSS

2. **Data preprocessing:** The data in the input excel file has the first column with data as 1, 2 or 3. This represents the type of the data. This column is useful in finding out the base for the different patterns as below.

   In order to do a proper data analysis to find out the related patterns by establishing relations in the data, the file is divided based on the type specified. The individual data after splitting is placed in individual sheets of a single excel file.

   Data in individual sheet is given a proper relevant column name considering the format and description given.

```
In [3]: type1=pd.read_excel("proj-data.xlsx",sheetname=0)
        type1.head()
```

Out[3]:

	Type	User	Machine	Date	LoginTime	LogoutTime	AvgUserProcess	MaxUserProcess	TotalKeyboadCharacters	CPUUse
0	1	U01	M01	90108	80010	170040	22	70	12345	12098
1	1	U03	M03	90108	82010	172040	22	70	12345	12098
2	1	U05	M05	90108	82310	171040	22	70	12345	12098
3	1	U07	M07	90108	80010	170040	22	70	12345	12098
4	1	U09	M09	90108	81040	170040	22	70	12345	12098

```
In [4]: type2=pd.read_excel("proj-data.xlsx",sheetname=1)
        type2.head()
```

Out[4]:

	Type	User	Machine	Date	StartTime	Program	ExecutionTime	File	Status	Printer	PagesPrinted
0	2	U01	M01	90108	90201	LP010	340	F0059	R	PR1	10.0
1	2	U03	M03	90108	90201	LP020	340	F0059	R	PR1	10.0
2	2	U05	M05	90108	90201	UP310	340	F0010	RW	PR2	10.0
3	2	U07	M07	90108	90201	LP010	340	F0010	RW	PR2	10.0
4	2	U09	M09	90108	90201	LP095	340	F0010	RW	PR2	10.0

```
In [5]: type3=pd.read_excel("proj-data.xlsx",sheetname=2)
        type3.head()
```

Out[5]:

	Type	User	Machine	Date	StartTime	EmailProgram	EmailAddress	Status	Bytes	Attachments
0	3	U01	M01	90108	120656	E1	jones@pqr.com	S	209003	1
1	3	U02	M02	90108	120656	E1	jones@pqr.com	S	209003	1
2	3	U03	M03	90108	120656	E1	jones@pqr.com	S	209003	1
3	3	U05	M05	90108	120656	E1	jones@pqr.com	S	209003	1
4	3	U07	M07	90108	120656	E1	smith@abc.org	S	209003	1

3. **Data cleaning:** Cleaning the data is essential to get the accurate results. Cleaning the data would involve tasks such as removing the null values, filling the null values with “NA”. As sometimes these null values would create a hurdle in getting the best possible accurate result, data cleaning plays a major role in building the best data model.

A. Check for any null values:

Type1 Data:

```
In [6]: type1.isnull().any()

Out[6]: Type                      False
        User                      False
        Machine                   False
        Date                      False
        LoginTime                 False
        LogoutTime                False
        AvgUserProcess            False
        MaxUserProcess            False
        TotalKeyboadCharacters    False
        CPUUse                    False
        dtype: bool
```

Type2 Data:

```
In [7]: type2.isnull ().any()

Out[7]: Type             False
        User             False
        Machine          False
        Date             False
        StartTime        False
        Program          False
        ExecutionTime    False
        File             False
        Status           False
        Printer           True
        PagesPrinted      True
        dtype: bool
```

Type3 Data:

```
In [165]: type3.isnull().any()

Out[165]: Type               False
          User               False
          Machine            False
          Date               False
          StartTime          False
          EmailProgram       False
          EmailAddress       False
          Status             False
          Bytes              False
          Attachments        False
          EmailAddressNum    False
          UserNum            False
          MachineNum         False
          dtype: bool
```

B. Dropping Null Values:

Shape of the type2**:** Rows: 385, Columns: 11

```
In [8]: print("shape",np.shape(type2))
        shape (385, 11)

In [9]: type2=type2.dropna()

In [10]: print("shape",np.shape(type2))
         shape (359, 11)
```

Shape after dropping null values: Rows:359, Columns: 11
No of rows deleted after dropping null values: 359-385 = 26

```
In [41]: type2.isnull().any()

Out[41]: Type             False
         User             False
         Machine          False
         Date             False
         StartTime        False
         Program          False
         ExecutionTime    False
         File             False
         Status           False
         Printer          False
         PagesPrinted     False
         dtype: bool
```

4. **Exploratory data analysis:** Here the columns User and Machine has the format ‘U01’ and ‘M01’. As the data value consists alphabets ‘U’ and ‘M’ in the columns, it would be difficult to carry the data analysis and perform certain metric functions. Creating new column “UserNum” and “MachineNum” so that these columns would be easily handled by various functions in python for the data analysis.

```
In [14]: type1['UserNum']=type1.User.str[1: ]

In [15]: type1.head()
```

Out[15]:

	Type	User	Machine	Date	LoginTime	LogoutTime	AvgUserProcess	MaxUserProcess	TotalKeyboadCharacters	CPUUse	UserNum
0	1	U01	M01	90108	80010	170040	22	70	12345	12098	01
1	1	U03	M03	90108	82010	172040	22	70	12345	12098	03
2	1	U05	M05	90108	82310	171040	22	70	12345	12098	05
3	1	U07	M07	90108	80010	170040	22	70	12345	12098	07
4	1	U09	M09	90108	81040	170040	22	70	12345	12098	09

```
In [16]: type1['MachineNum']=type1.Machine.str[1: ]

In [17]: type1.head()
```

Out[17]:

	Type	User	Machine	Date	LoginTime	LogoutTime	AvgUserProcess	MaxUserProcess	TotalKeyboadCharacters	CPUUse	UserNum	MachineNum
0	1	U01	M01	90108	80010	170040	22	70	12345	12098	01	01
1	1	U03	M03	90108	82010	172040	22	70	12345	12098	03	03
2	1	U05	M05	90108	82310	171040	22	70	12345	12098	05	05
3	1	U07	M07	90108	80010	170040	22	70	12345	12098	07	07
4	1	U09	M09	90108	81040	170040	22	70	12345	12098	09	09

Type1 data:
Describing the type1 data:

```
In [169]: type1.describe()
```

Out[169]:

	Type	Date	LoginTime	LogoutTime	AvgUserProcess	MaxUserProcess	TotalKeyboadCharacters	CPUUse
count	389.0	389.000000	389.000000	389.000000	389.000000	389.000000	389.000000	389.000000
mean	1.0	91516.483290	97588.637532	177117.069409	16.953728	47.748072	11213.907455	10973.136247
std	0.0	883.237375	34737.775191	15015.610113	6.334972	13.541204	5739.709564	5451.201059
min	1.0	90108.000000	51010.000000	152940.000000	9.000000	30.000000	1210.000000	1352.000000
25%	1.0	90808.000000	80540.000000	171300.000000	10.000000	42.000000	12345.000000	12098.000000
50%	1.0	91608.000000	82010.000000	172340.000000	15.000000	45.000000	12345.000000	13400.000000
75%	1.0	92308.000000	110540.000000	172940.000000	22.000000	55.000000	15455.000000	13558.000000
max	1.0	93008.000000	182540.000000	231300.000000	25.000000	70.000000	17455.000000	17234.000000

Information of type1 data:

```
In [170]: type1.info()
```

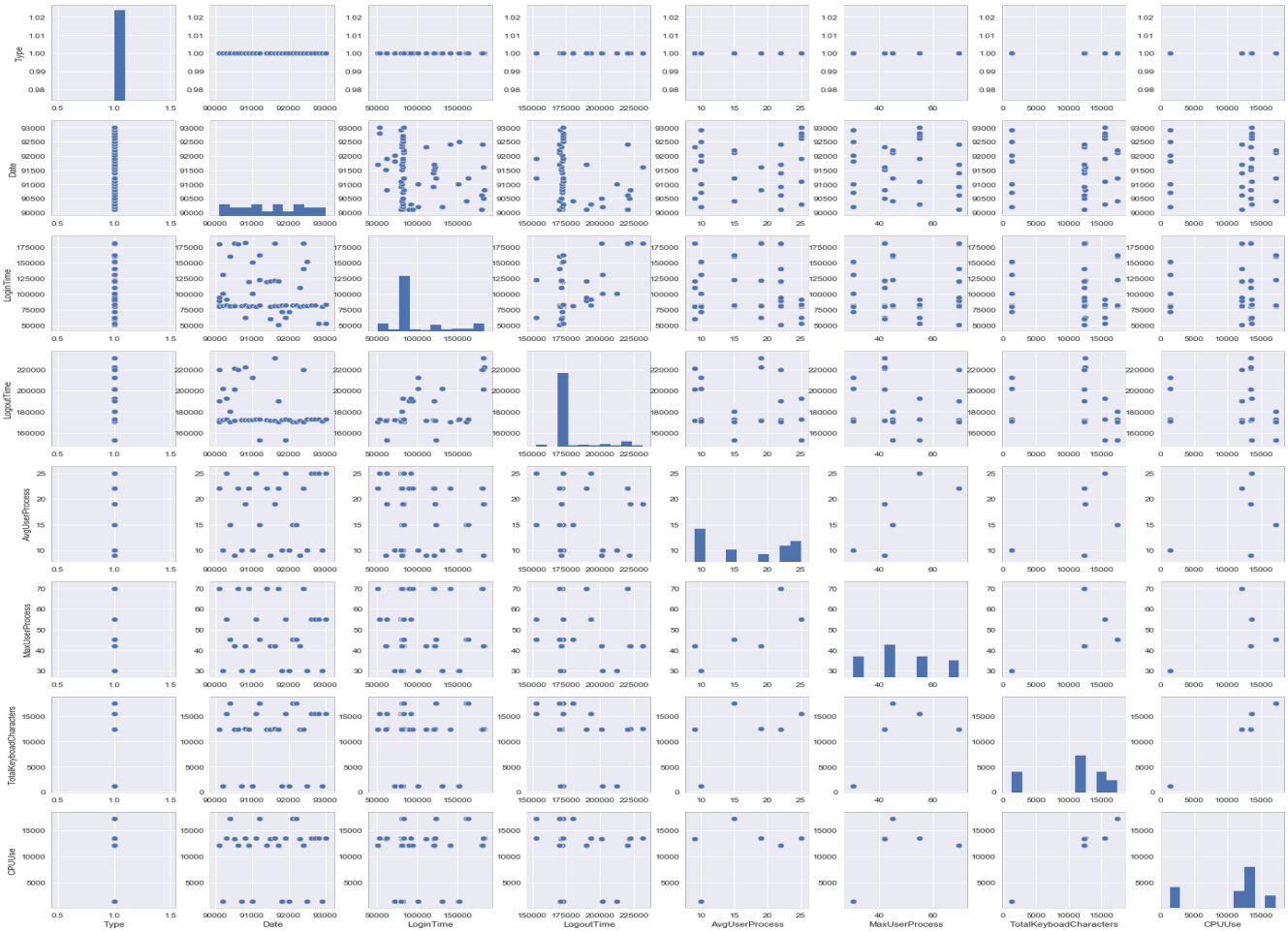
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 389 entries, 0 to 388
Data columns (total 13 columns):
Type                        389 non-null int64
User                        389 non-null object
Machine                     389 non-null object
Date                        389 non-null int64
LoginTime                   389 non-null int64
LogoutTime                  389 non-null int64
AvgUserProcess              389 non-null int64
MaxUserProcess              389 non-null int64
TotalKeyboadCharacters      389 non-null int64
CPUUse                      389 non-null int64
UserNum                     389 non-null object
MachineNum                  389 non-null object
DateString                  389 non-null object
dtypes: int64(8), object(5)
memory usage: 39.6+ KB
```

User information of type1 data:

```
In [175]: type1.User.describe()
```

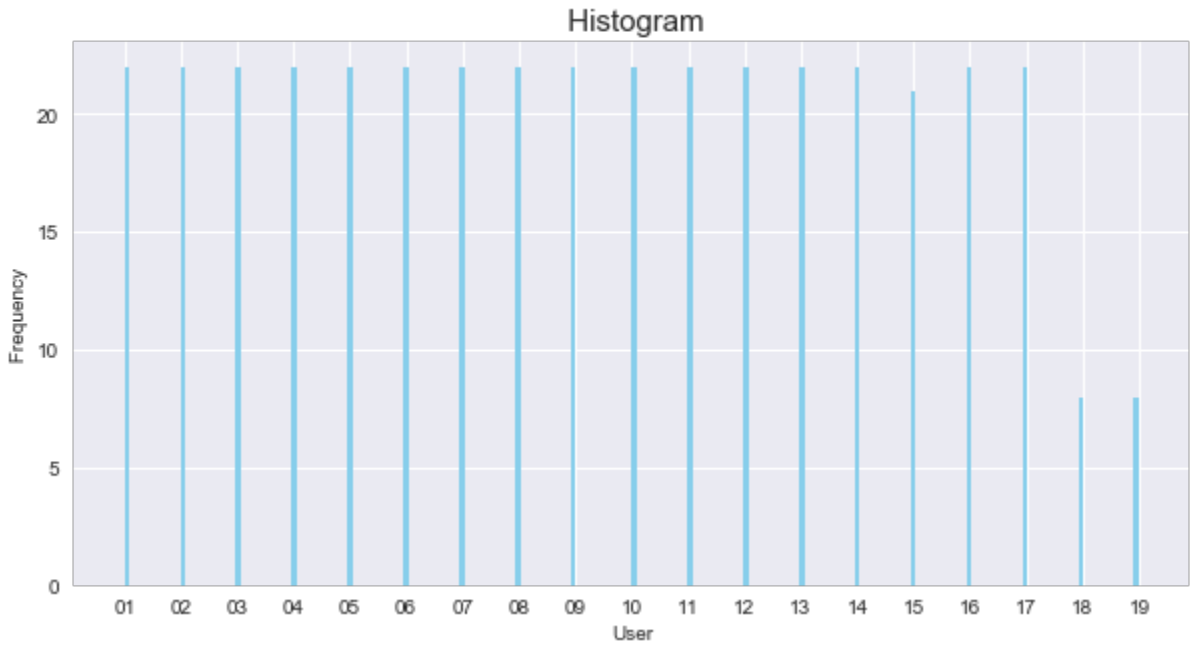
```
Out[175]: count     389
          unique     19
          top       U08
          freq       22
          Name: User, dtype: object
```

Pair plot: Visualizing the type1 data with a pair plot considering all the attributes.



User: Histogram: User information could be analyzed primarily with a histogram. A histogram shows the frequency count with respect to individual user.

```
In [18]: fig, ax = plt.subplots(1, 1, figsize = (10, 5))
         ax.hist(type1.UserNum, bins = 200, range = [min(type1.UserNum), max(type1.UserNum)], label = "price",color = "skyblue")
         ax.set_title("\n \n  Histogram ", fontsize = 15)
         ax.set_xlabel("User", fontsize = 10)
         ax.set_ylabel(" Frequency ", fontsize = 10)
         plt.show()
```

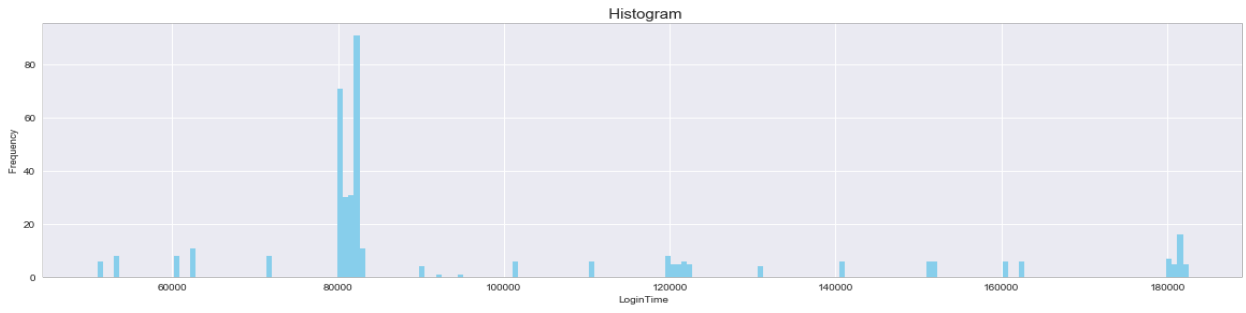


Here, User 18 and 19 has less frequency.

1. LOGIN ACCESS PATTERN:

LoginTime: Histogram:

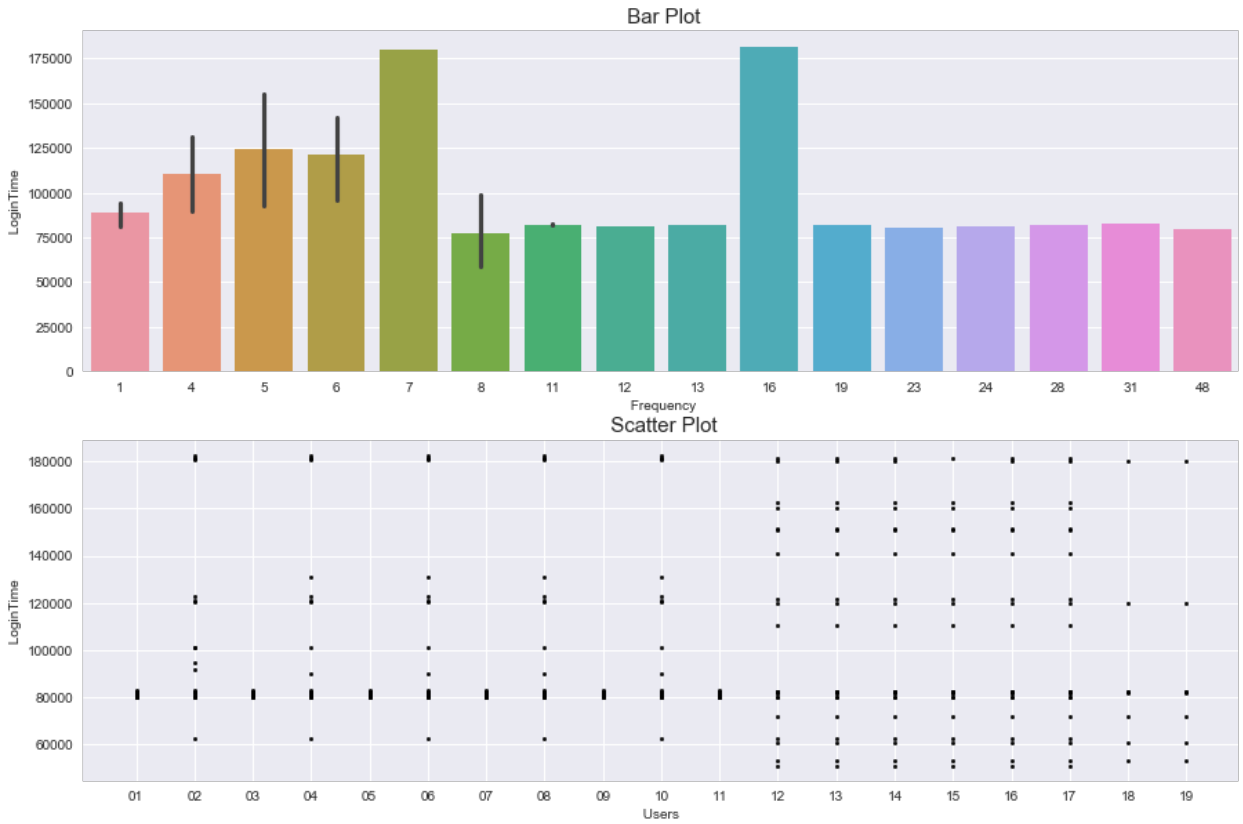
```
In [20]: fig, ax = plt.subplots(1, 1, figsize = (20, 5))
         ax.hist(type1.LoginTime, bins = 200, range = [min(type1.LoginTime), max(type1.LoginTime)], label = "price",color = "skyl
         ax.set_title("\n \n  Histogram ", fontsize = 15)
         ax.set_xlabel("LoginTime", fontsize = 10)
         ax.set_ylabel(" Frequency ", fontsize = 10)
         plt.show()
```



LoginTime: Bar plot and Scatter plot: Bar plots are used for discrete counts and scatter plots are used for visualize relation between two quantitative attributes.

In [21]:
```
logintime = type1["LoginTime"].value_counts()
print("Unique Login Times :", logintime.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 10))
sns.barplot(logintime.values, logintime.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type1['UserNum'].values
f2 = type1['LoginTime'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" LoginTime ", fontsize = 10)
ax[1].set_xlabel(" Users ", fontsize = 10)
ax[1].set_ylabel(" LoginTime ", fontsize = 10)
plt.show()
```

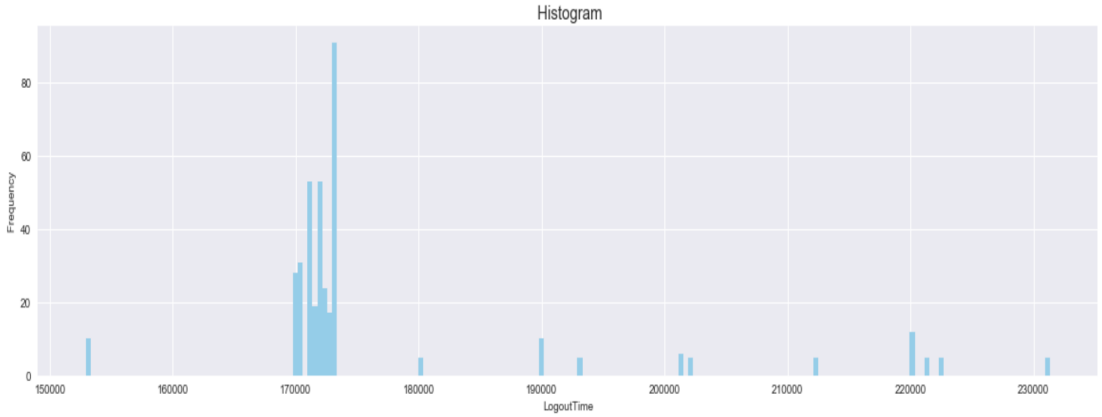
```
Unique Login Times : 39
```



From the above scatter plot, we infer that every user logs in at 8:00 AM.

LogoutTime: Histogram:

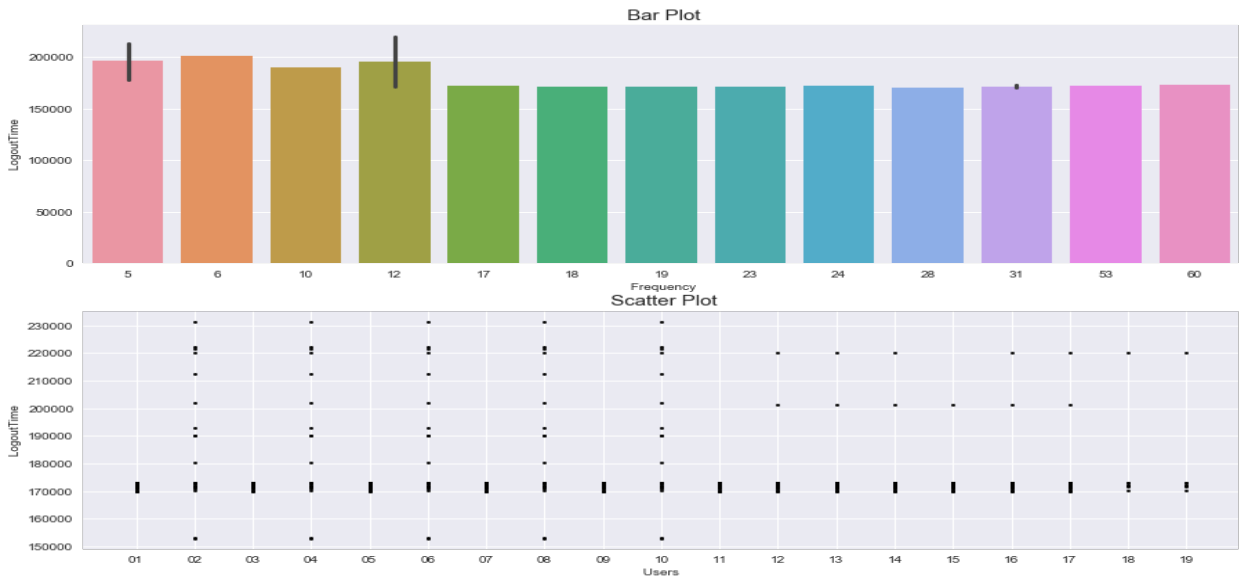
```
In [22]: fig, ax = plt.subplots(1, 1, figsize = (20, 5))
ax.hist(type1.LogoutTime, bins = 200, range = [min(type1.LogoutTime), max(type1.LogoutTime)], label = "price",color = "
ax.set_title("\n \n  Histogram ", fontsize = 15)
ax.set_xlabel("LogoutTime", fontsize = 10)
ax.set_ylabel(" Frequency ", fontsize = 10)
plt.show()
```



Bar plot and scatter plot:

```
In [23]: logouttime = type1["LogoutTime"].value_counts()
print("Unique Logout Times :", logouttime.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 10))
sns.barplot(logouttime.values, logouttime.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type1['UserNum'].values
f2 = type1['LogoutTime'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" LogoutTime ", fontsize = 10)
ax[1].set_xlabel(" Users ", fontsize = 10)
ax[1].set_ylabel(" LogoutTime ", fontsize = 10)
plt.show()
```

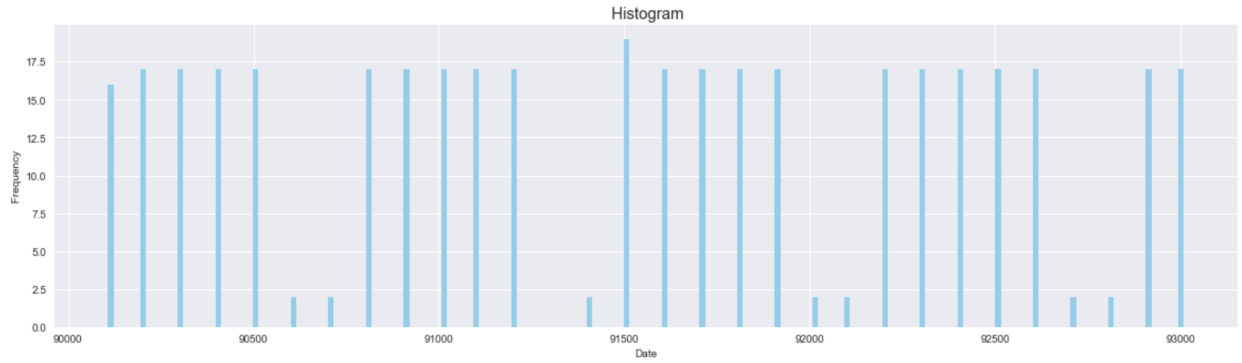
```
Unique Logout Times : 23
```



From the above scatter plot, we infer that every user logs out at 17:00 PM

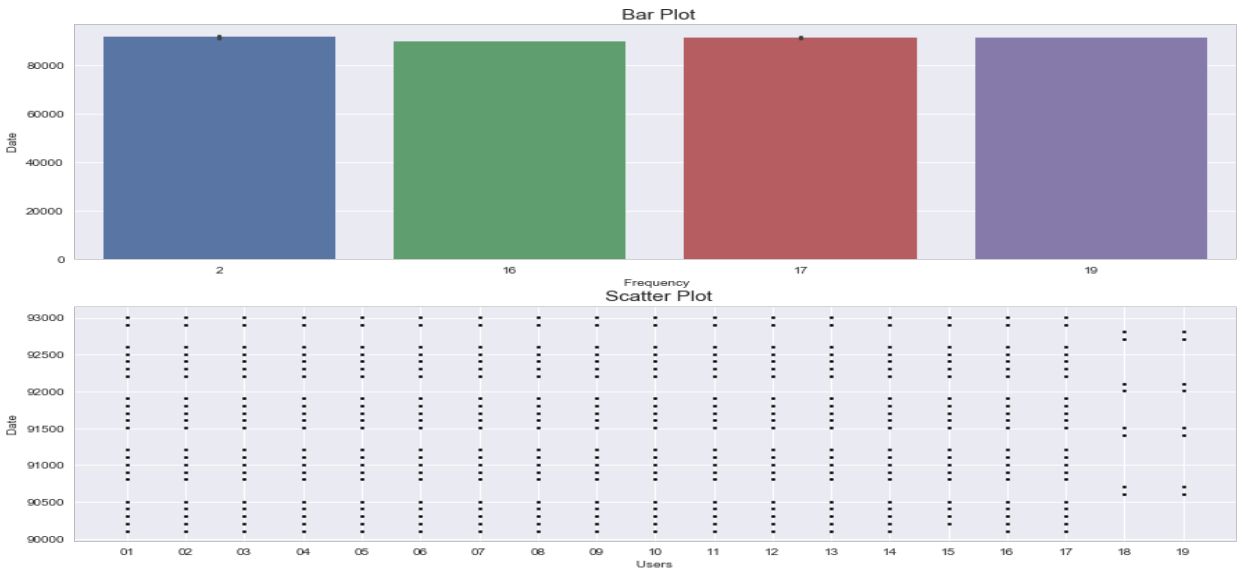
Date: Histogram:

```
In [25]: fig, ax = plt.subplots(1, 1, figsize = (20, 5))
ax.hist(type1.Date, bins = 200, range = [min(type1.Date), max(type1.Date)], label = "price",color = "skyblue")
ax.set_title("\n \n  Histogram ", fontsize = 15)
ax.set_xlabel("Date", fontsize = 10)
ax.set_ylabel(" Frequency ", fontsize = 10)
plt.show()
```



Bar pot and scatter plot:

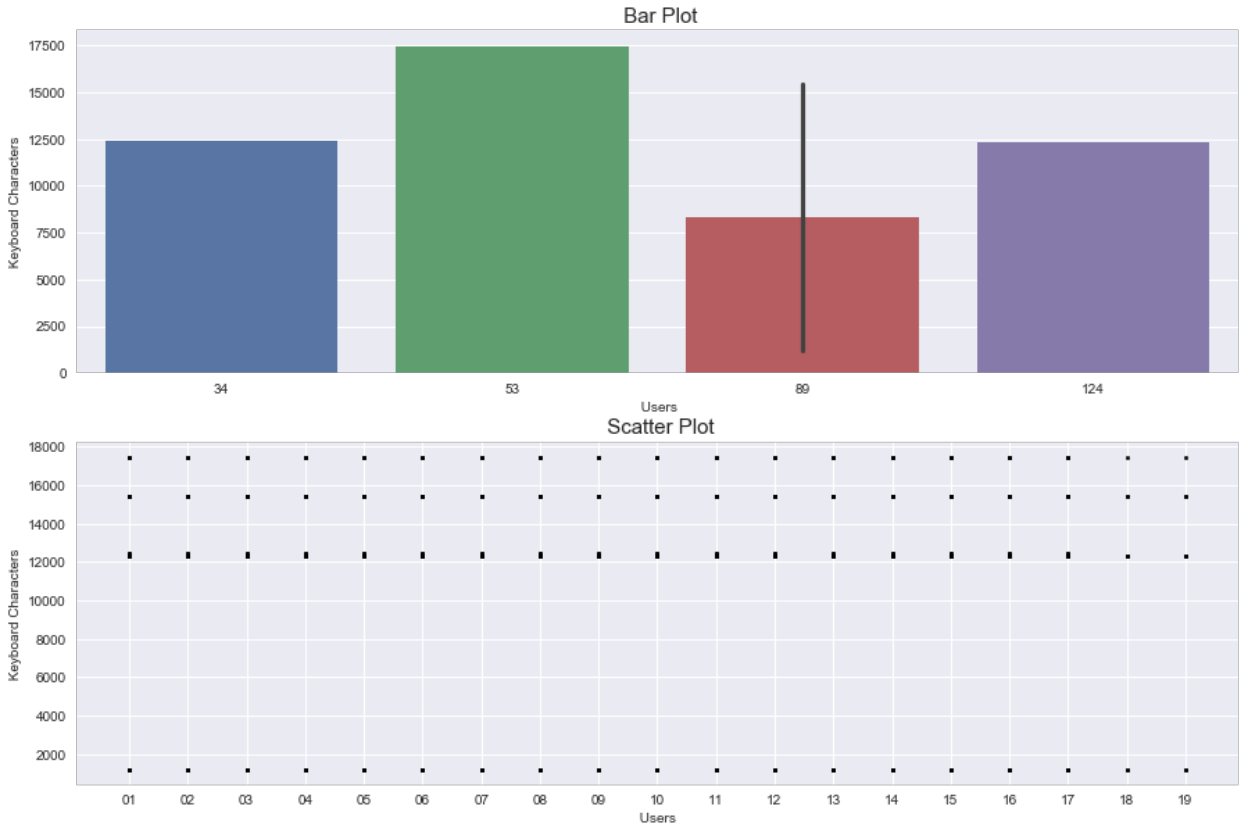
```
In [178]: dates = type1["Date"].value_counts()
print("Unique Brand Names :", dates.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 10))
sns.barplot(dates[:].values, dates[:].index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type1['UserNum'].values
f2 = type1['Date'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" Date ", fontsize = 10)
ax[1].set_xlabel(" Users ", fontsize = 10)
ax[1].set_ylabel(" Date ", fontsize = 10)
plt.show()
```



Here, we could see a different behaviour for users 18 and 19 on specific dates.

TotalKeyboadCharacters: Bar pot and scatter plot:

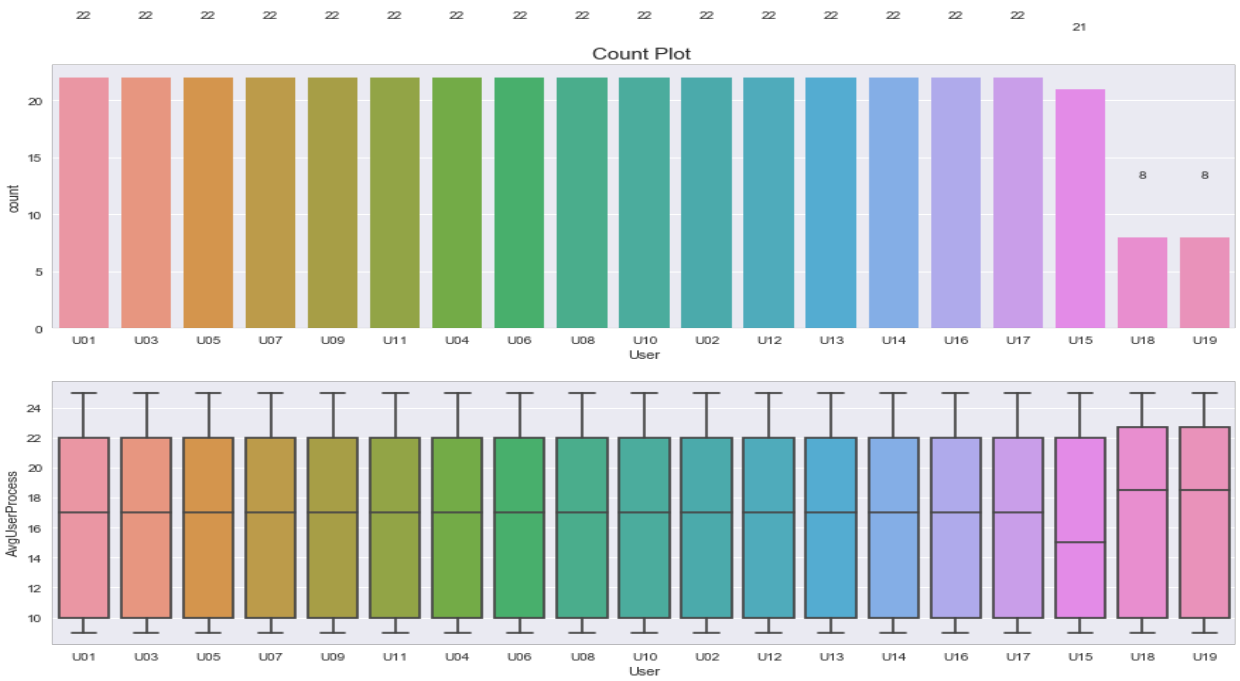
```
In [28]: KeyboardChar = type1["TotalKeyboadCharacters"].value_counts()
         print("Unique Keyboard Characters :", KeyboardChar.size)
         fig, ax = plt.subplots(2, 1, figsize = (15, 10))
         sns.barplot(KeyboardChar[:].values, KeyboardChar[:].index,ax = ax[0])
         ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
         ax[0].set_xlabel(" Sale Price", fontsize = 10)
         f1 = type1['UserNum'].values
         f2 = type1['TotalKeyboadCharacters'].values
         X = np.array(list(zip(f1, f2)))
         ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
         ax[1].scatter(f1, f2, c='black', s=7)
         ax[0].set_xlabel(" Users", fontsize = 10)
         ax[0].set_ylabel(" Keyboard Characters ", fontsize = 10)
         ax[1].set_xlabel(" Users ", fontsize = 10)
         ax[1].set_ylabel(" Keyboard Characters ", fontsize = 10)
         plt.show()

         Unique Keyboard Characters : 5
```



Here, we could find the same data distribution for all the users.

AvgUserProcess:Count plot and Box plot:

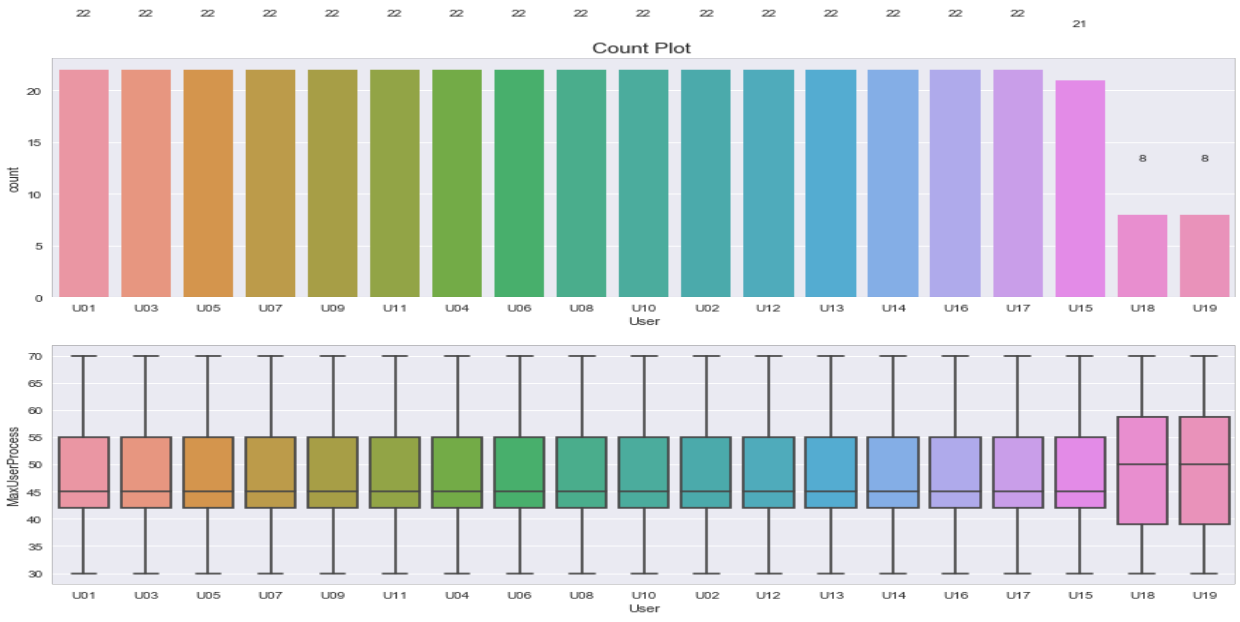
```
In [29]: fig, ax = plt.subplots(2, 1, figsize = (15,10))
         sns.countplot(type1.User, ax = ax[0])
         rectangles = ax[0].patches
         ax[0].set_title("Count Plot ", fontsize = 15)
         labels = type1.User.value_counts().values
         for rect, label in zip(rectangles, labels):
             height = rect.get_height()
             ax[0].text(rect.get_x() + rect.get_width()/2, height + 5, label, ha = "center", va = "bottom")
         sns.boxplot(x = type1.User, y = type1.AvgUserProcess,showfliers = False, orient = "v", ax = ax[1])
         plt.show()
```



Here, we could observe almost similar behaviour for all the users.

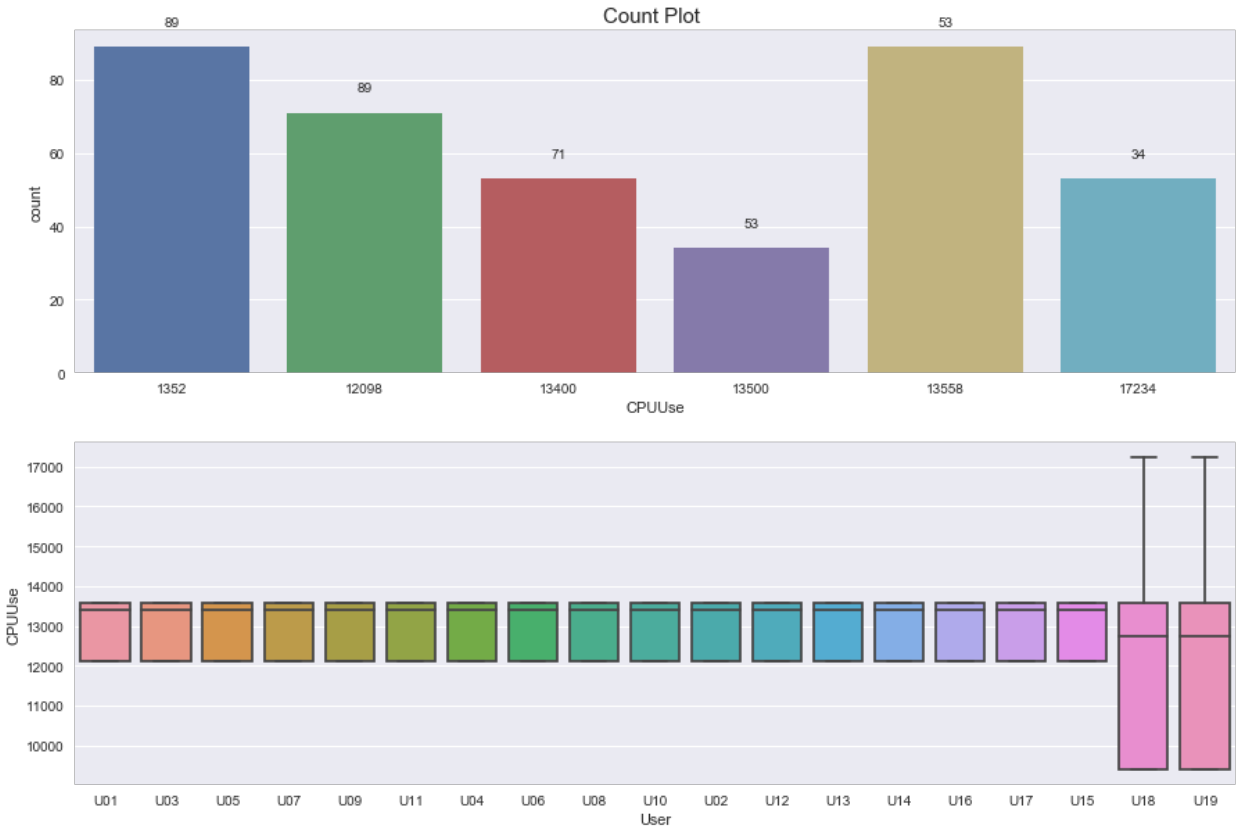
MaxUserProcess: Count plot and box plot

In [30]:
```
fig, ax = plt.subplots(2, 1, figsize = (15,10))
sns.countplot(type1.User, ax = ax[0])
rectangles = ax[0].patches
ax[0].set_title("Count Plot ", fontsize = 15)
labels = type1.User.value_counts().values
for rect, label in zip(rectangles, labels):
    height = rect.get_height()
    ax[0].text(rect.get_x() + rect.get_width()/2, height + 5, label, ha = "center", va = "bottom")
sns.boxplot(x = type1.User, y = type1.MaxUserProcess,showfliers = False, orient = "v", ax = ax[1])
plt.show()
```



Here, we could notice similar behaviour for all Users except 18 and 19 and they have max usage processes.
CPUUse: count plot and box plot

```
In [31]: fig, ax = plt.subplots(2, 1, figsize = (15,10))
sns.countplot(type1.CPUUse, ax = ax[0])
rectangles = ax[0].patches
ax[0].set_title("Count Plot ", fontsize = 15)
labels = type1.CPUUse.value_counts().values
for rect, label in zip(rectangles, labels):
    height = rect.get_height()
    ax[0].text(rect.get_x() + rect.get_width()/2, height + 5, label, ha = "center", va = "bottom")
sns.boxplot(x = type1.User, y = type1.CPUUse,showfliers = False, orient = "v", ax = ax[1])
plt.show()
```

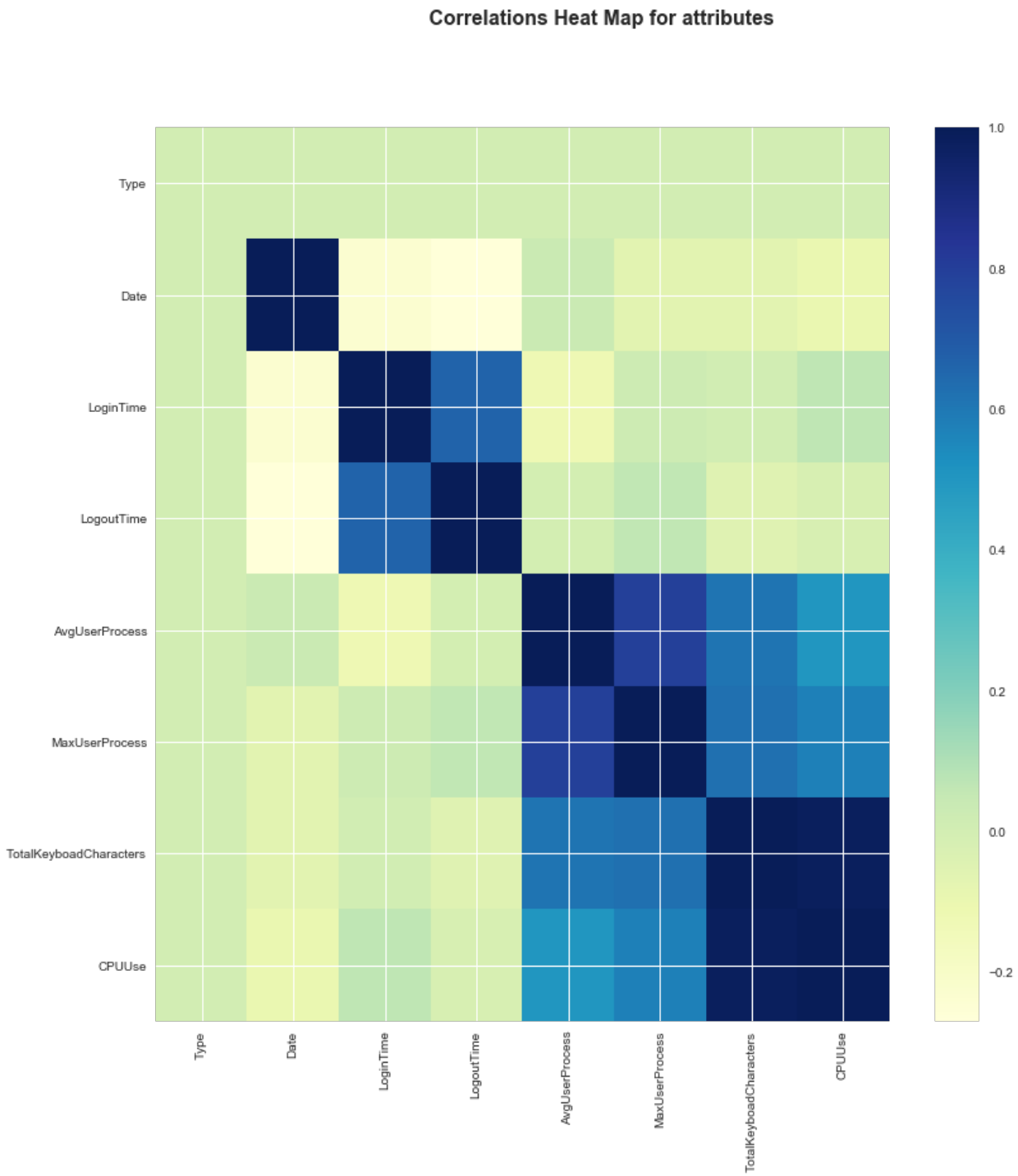


Here, we could notice U18 and U19 has highest CPU usage.
Statistics: **Correlation:** It is a numeric measure of statistical measure between attributes.

```
In [32]: corltn=type1.corr()
corltn=corltn.fillna(0)
plt.figure(figsize=(13, 13))
plt.imshow(corltn, cmap='YlGnBu', interpolation='none', aspect='auto')
plt.colorbar()
plt.xticks(range(len(corltn)), corltn.columns, rotation='vertical')
plt.yticks(range(len(corltn)), corltn.columns);
plt.suptitle(' Correlations Heat Map for attributes', fontsize=16, fontweight='bold')
```

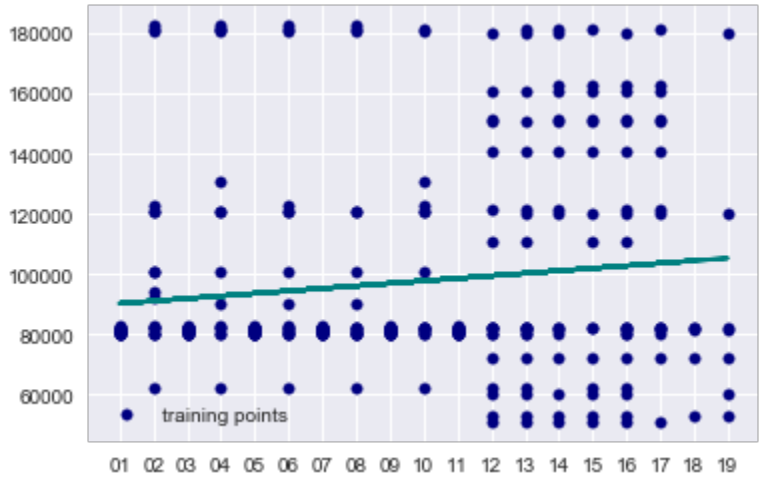
```
Out[32]: Text(0.5,0.98,' Correlations Heat Map for attributes')
```

Heat Map: Visualization of correlation among the attributes with a heatmap.


Correlations Heat Map for attributes
Type
Date
LoginTime
LogoutTime
AvgUserProcess
MaxUserProcess
TotalKeyboadCharacters
CPUUse
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

5. **Build data model: Linear Regression:** Linear Regression is used to express the mathematical relationship between attributes to predict pattern.

```
In [33]: X_train, x_test, Y_train, y_test = train_test_split(
         type1['UserNum'], type1['LoginTime'], test_size=0.2, random_state=42)
         regr = linear_model.LinearRegression()
         X_train = X_train[:, np.newaxis]
         Y_train = Y_train[:, np.newaxis]
         x_test = x_test[:, np.newaxis]
         regr.fit(X_train, Y_train)
         y_predict = regr.predict(x_test)
         plt.scatter(X_train, Y_train, color='navy', s=30, marker='o', label="training points")
         plt.plot(x_test, y_predict, color='teal', linewidth=2)
         plt.legend(loc='best')
         plt.show()
```



Performance Evaluation: Calculating the mean square error for the applied linear regression model.

```
In [34]: r_data= type1[['UserNum','MachineNum','LogoutTime']]
         X_train, x_test, Y_train, y_test = train_test_split(
         r_data, type1['LoginTime'], test_size=0.2, random_state=42)
         regr = linear_model.LinearRegression()
         Y_train = Y_train[:, np.newaxis]
         regr.fit(X_train, Y_train)
         y_predict = regr.predict(x_test)
         mean_squared_error(y_test,y_predict)

Out[34]: 537312977.82789862

In [35]: r_data= type1[['Date','LoginTime','LogoutTime']]
         X_train, x_test, Y_train, y_test = train_test_split(
         r_data, type1['UserNum'], test_size=0.2, random_state=42)
         regr = linear_model.LinearRegression()
         Y_train = Y_train[:, np.newaxis]
         regr.fit(X_train, Y_train)
         y_predict = regr.predict(x_test)
         mean_squared_error(y_test,y_predict)

Out[35]: 28.317679623303793

In [36]: r_data= type1[['Date','LoginTime','LogoutTime','MachineNum','AvgUserProcess','MaxUserProcess']]
         X_train, x_test, Y_train, y_test = train_test_split(
         r_data, type1['UserNum'], test_size=0.2, random_state=42)
         regr = linear_model.LinearRegression()
         Y_train = Y_train[:, np.newaxis]
         regr.fit(X_train, Y_train)
         y_predict = regr.predict(x_test)
         mean_squared_error(y_test,y_predict)

Out[36]: 9.8281541853993435
```

Here, we achievent a fair score of mean square and we could consider this model for finding patterns.

2. **PROGRAM ACCESS PATTERN**:

Describing the type2 data:

```
In [180]: type2.describe()
```

Out[180]:

	Type	Date	StartTime	ExecutionTime	PagesPrinted	ProgramNum	FileNum	StatusNum	PrinterNum
count	359.0	359.000000	359.000000	359.000000	359.000000	359.000000	359.000000	359.000000	359.000000
mean	2.0	91359.810585	127702.114206	792.980501	7.175487	12.573816	9.100279	0.420613	1.523677
std	0.0	1046.217346	28201.373563	569.290235	5.680001	6.937026	7.669207	0.494346	1.385831
min	2.0	90108.000000	83901.000000	340.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	2.0	90308.000000	100901.000000	340.000000	3.000000	6.500000	2.000000	0.000000	0.500000
50%	2.0	91108.000000	125201.000000	450.000000	5.000000	15.000000	9.000000	0.000000	1.000000
75%	2.0	92208.000000	145201.000000	1040.000000	10.000000	19.000000	14.000000	1.000000	3.000000
max	2.0	93008.000000	201251.000000	2640.000000	21.000000	23.000000	28.000000	1.000000	5.000000

Pair Plot:

Information of type2 data:

```
In [181]: type2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 359 entries, 0 to 381
Data columns (total 17 columns):
Type              359 non-null int64
User              359 non-null object
Machine           359 non-null object
Date              359 non-null int64
StartTime         359 non-null int64
Program           359 non-null object
ExecutionTime     359 non-null int64
File              359 non-null object
Status            359 non-null object
Printer           359 non-null object
PagesPrinted      359 non-null float64
UserNum           359 non-null object
MachineNum        359 non-null object
ProgramNum        359 non-null int64
FileNum           359 non-null int64
StatusNum         359 non-null int64
PrinterNum        359 non-null int64
dtypes: float64(1), int64(8), object(8)
memory usage: 60.5+ KB
```

Column conversion: Adding new columns so as to change the User and Machine columns to a numeric data value.

```
In [39]: type2['UserNum']=type2.User.str[1: ]
```

```
In [40]: type2['MachineNum']=type2.Machine.str[1: ]
```

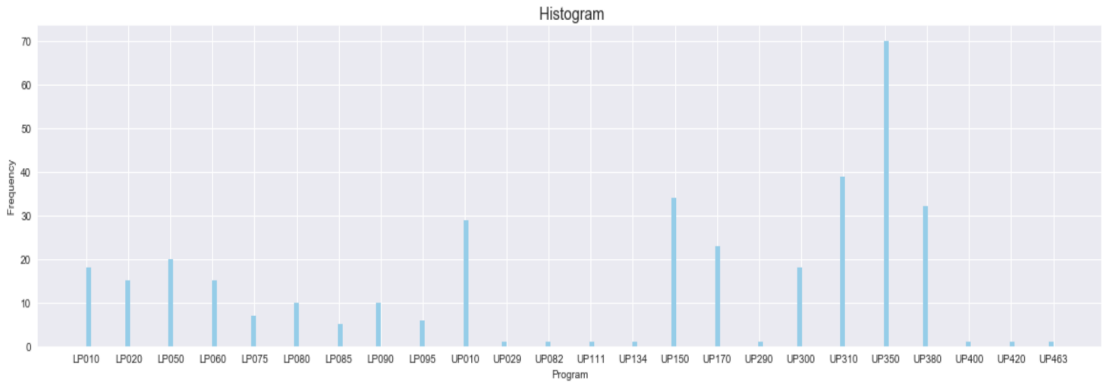
```
In [41]: type2.head()
```

Out[41]:

	Type	User	Machine	Date	StartTime	Program	ExecutionTime	File	Status	Printer	PagesPrinted	UserNum	MachineNum
0	2	U01	M01	90108	90201	LP010	340	F0059	R	PR1	10.0	01	01
1	2	U03	M03	90108	90201	LP020	340	F0059	R	PR1	10.0	03	03
2	2	U05	M05	90108	90201	UP310	340	F0010	RW	PR2	10.0	05	05
3	2	U07	M07	90108	90201	LP010	340	F0010	RW	PR2	10.0	07	07
4	2	U09	M09	90108	90201	LP095	340	F0010	RW	PR2	10.0	09	09

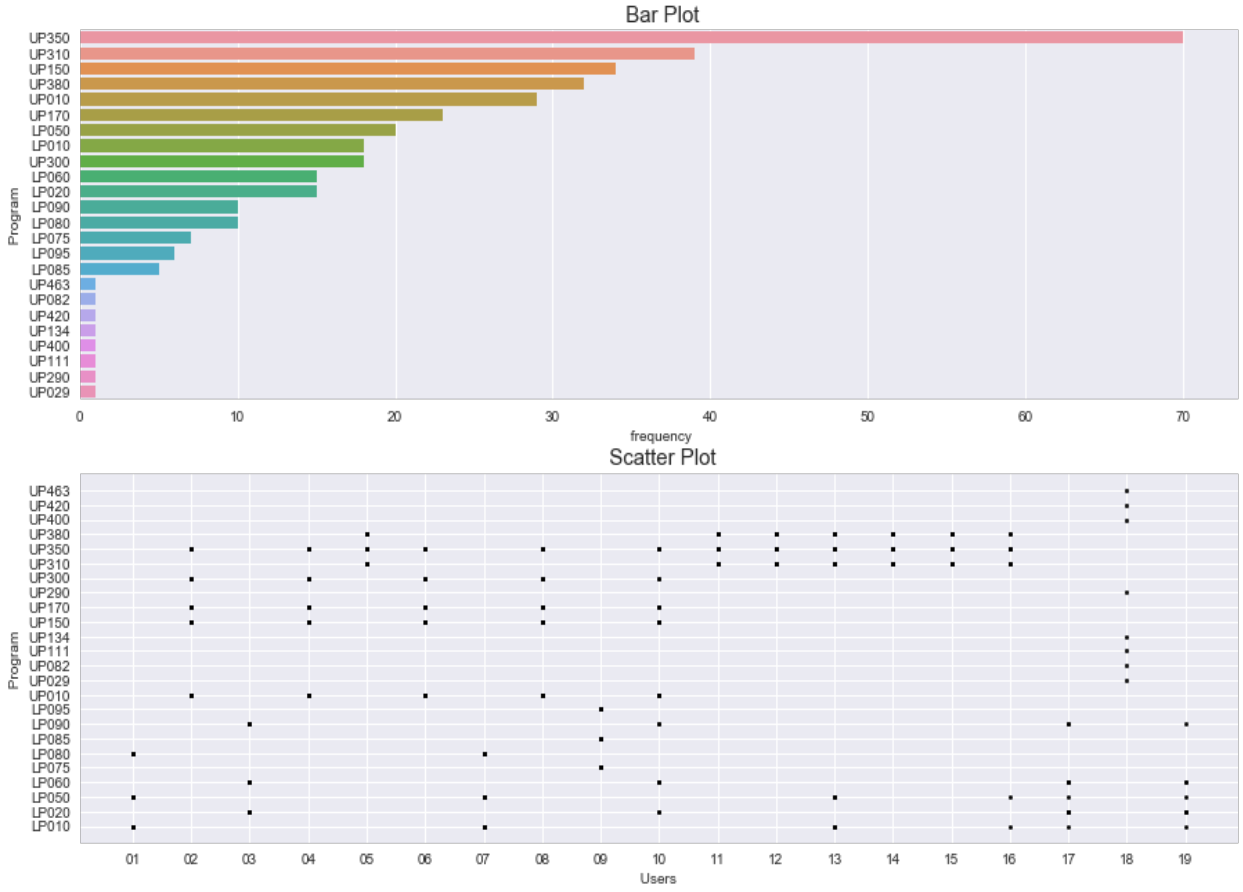
Program: Histogram:

```
In [42]: fig, ax = plt.subplots(1, 1, figsize = (20, 5))
         ax.hist(type2.Program, bins = 200, range = [min(type2.Program), max(type2.Program)], label = "price",color = "skyblue")
         ax.set_title("\n \n  Histogram ", fontsize = 15)
         ax.set_xlabel("Program", fontsize = 10)
         ax.set_ylabel(" Frequency ", fontsize = 10)
         plt.show()
```



Bar plot and Scatter plot:

```
In [43]: program = type2["Program"].value_counts()
         print("Unique Program Names :", program.size)
         fig, ax = plt.subplots(2, 1, figsize = (15, 10))
         sns.barplot(program.values, program.index,ax = ax[0])
         ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
         ax[0].set_xlabel(" program ", fontsize = 10)
         f1 = type2['UserNum'].values
         f2 = type2['Program'].values
         ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
         ax[1].scatter(f1, f2, c='black', s=7)
         ax[0].set_xlabel(" frequency", fontsize = 10)
         ax[0].set_ylabel(" Program ", fontsize = 10)
         ax[1].set_xlabel(" Users ", fontsize = 10)
         ax[1].set_ylabel(" Program ", fontsize = 10)
         plt.show()

Unique Program Names : 24
```

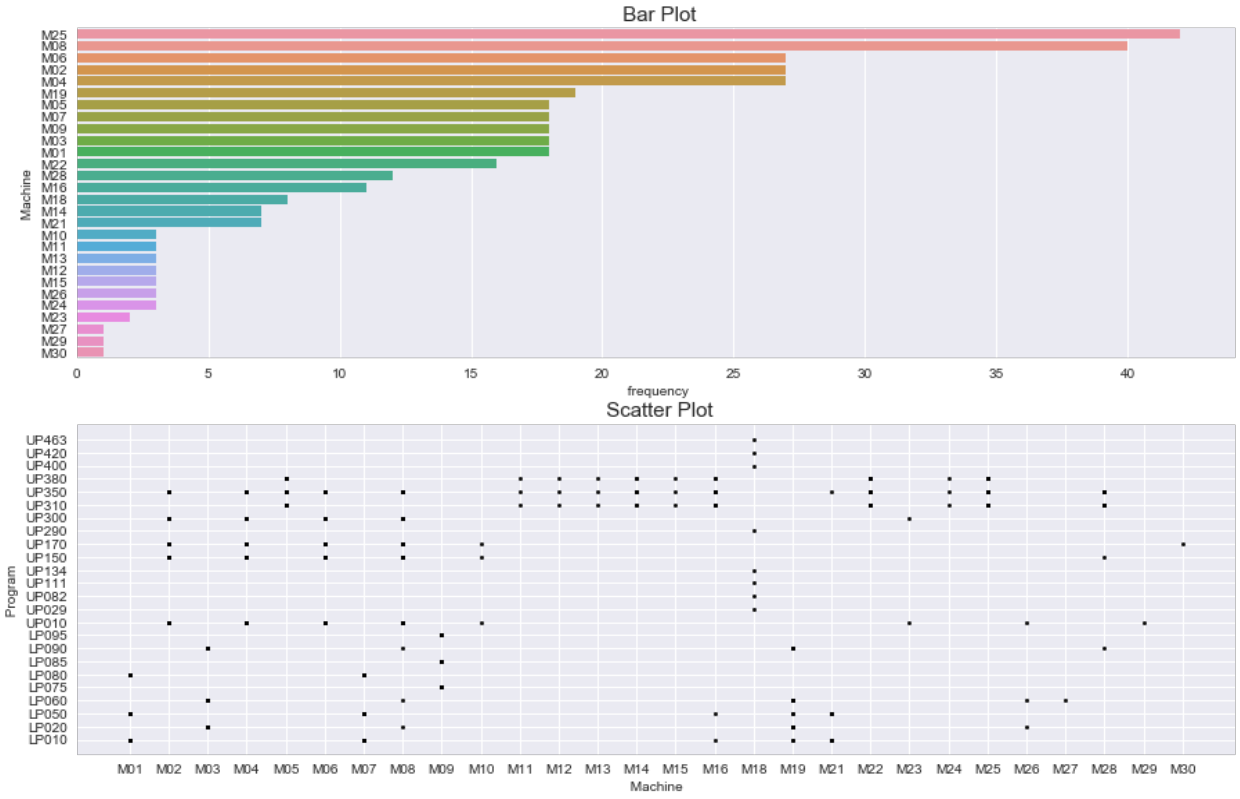


Here, we could notice UP350 has the highest frequency count.

Machine: Bar plot and scatter plot:

```
In [182]: machine2 = type2["Machine"].value_counts()
          print("Unique Machine Names :", machine2.size)
          fig, ax = plt.subplots(2, 1, figsize = (15, 10))
          sns.barplot(machine2.values, machine2.index,ax = ax[0])
          ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
          ax[0].set_xlabel(" User ", fontsize = 10)
          f1 = type2['Machine'].values
          f2 = type2['Program'].values
          ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
          ax[1].scatter(f1, f2, c='black', s=7)
          ax[0].set_xlabel(" frequency", fontsize = 10)
          ax[0].set_ylabel(" Machine ", fontsize = 10)
          ax[1].set_xlabel(" Machine ", fontsize = 10)
          ax[1].set_ylabel(" Program ", fontsize = 10)
          plt.show()
```

```
Unique Machine Names : 28
```

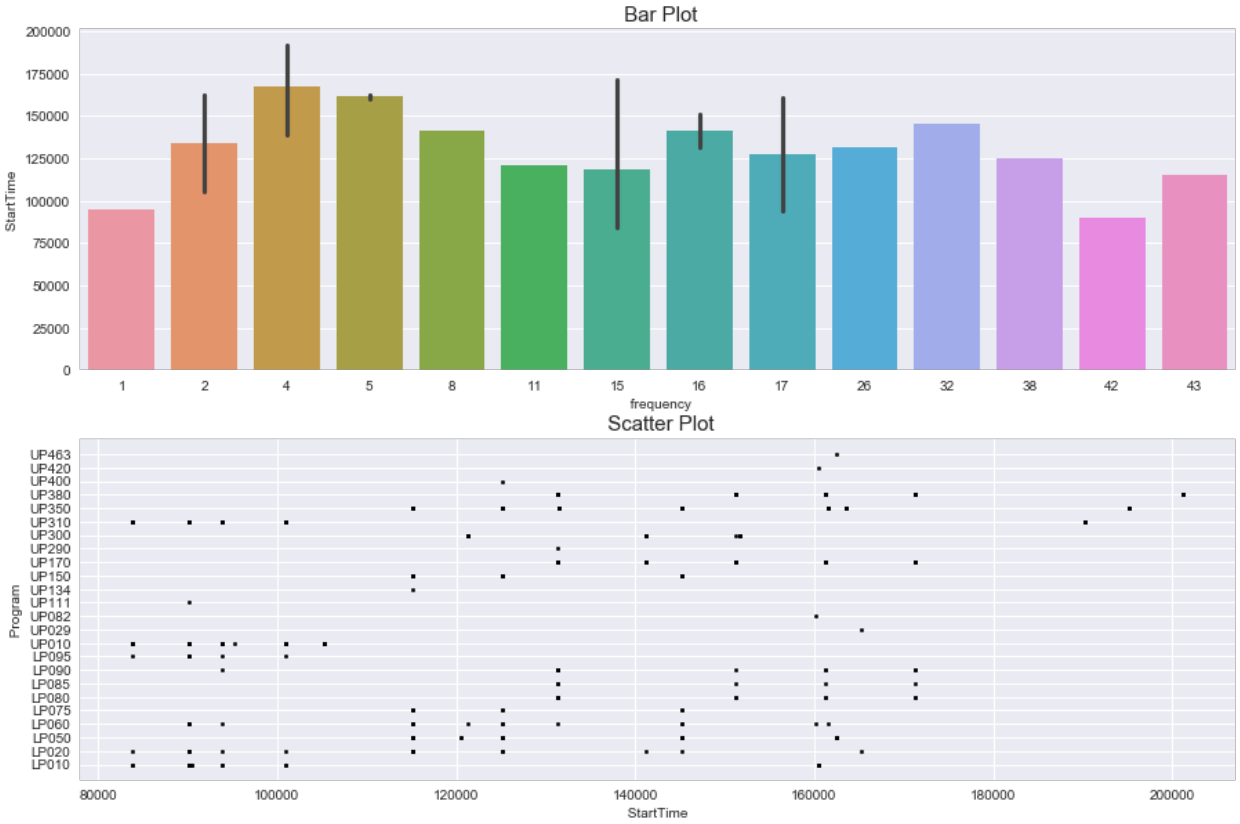


Here, we could see a similar patter for M11 to M15 with respect to program,

StartTime: Bar plot and scatter plot:

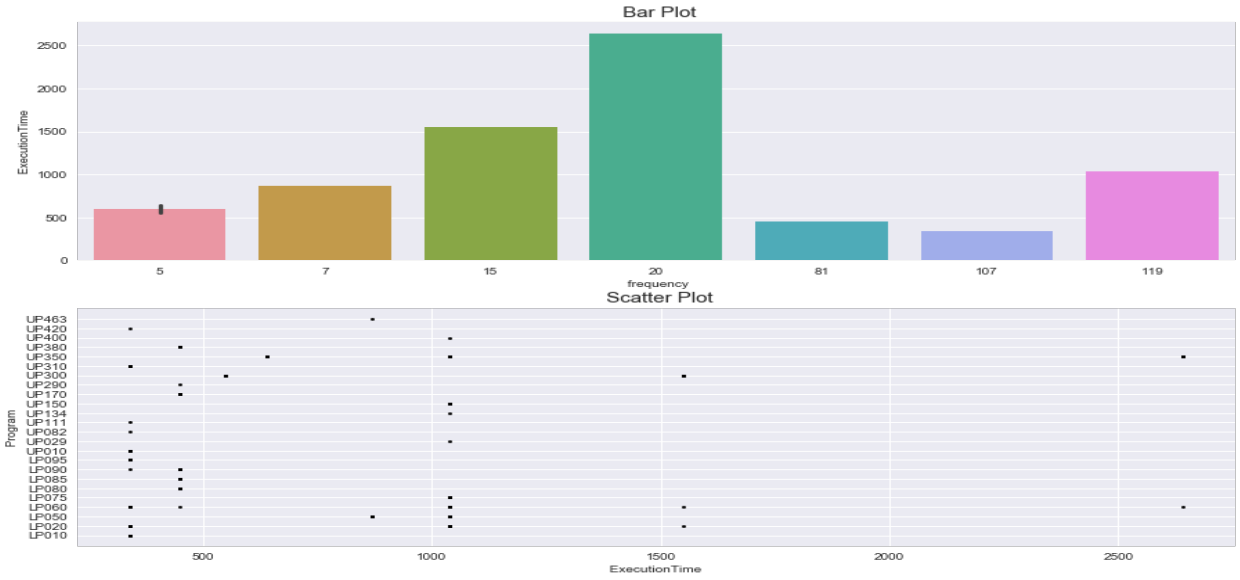
```
In [183]: starttime2 = type2["StartTime"].value_counts()
          print("Unique Start time:", starttime2.size)
          fig, ax = plt.subplots(2, 1, figsize = (15, 10))
          sns.barplot(starttime2.values, starttime2.index,ax = ax[0])
          ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
          ax[0].set_xlabel(" User ", fontsize = 10)
          f1 = type2['StartTime'].values
          f2 = type2['Program'].values
          ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
          ax[1].scatter(f1, f2, c='black', s=7)
          ax[0].set_xlabel(" frequency", fontsize = 10)
          ax[0].set_ylabel(" StartTime ", fontsize = 10)
          ax[1].set_xlabel(" StartTime ", fontsize = 10)
          ax[1].set_ylabel(" Program ", fontsize = 10)
          plt.show()
```

```
Unique Start time: 28
```



Here, we could find evenly distributed data and very rare for data beyong 180000 executiontime.

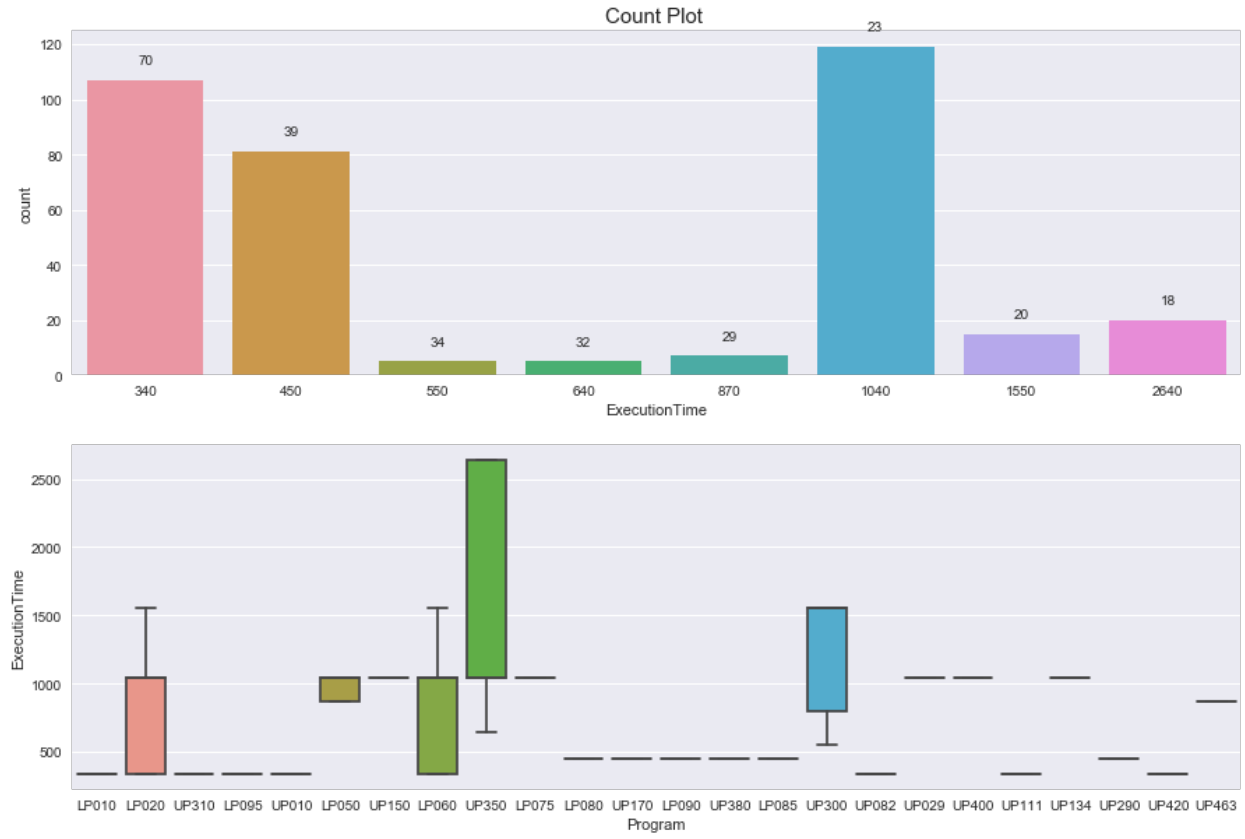
ExecutionTime: Bar plot and scatter plot:



Here, we could see data is distributed more below 500 execution time for the programs.

ExecutionTime and Program: Count plot anf box plot:

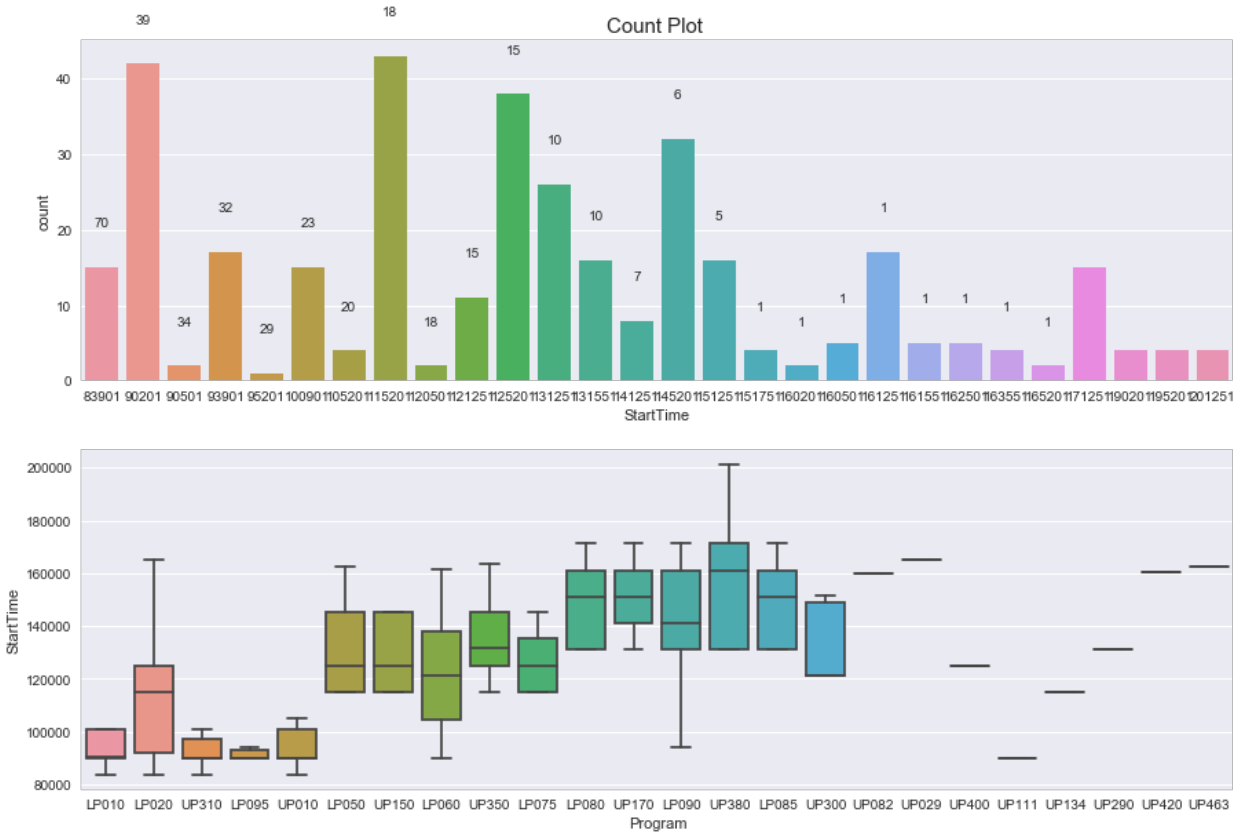
In [48]:
```
fig, ax = plt.subplots(2, 1, figsize = (15,10))
sns.countplot(type2.ExecutionTime, ax = ax[0])
rectangles = ax[0].patches
ax[0].set_title("Count Plot ", fontsize = 15)
labels = type2.Program.value_counts().values
for rect, label in zip(rectangles, labels):
    height = rect.get_height()
    ax[0].text(rect.get_x() + rect.get_width()/2, height + 5, label, ha = "center", va = "bottom")
sns.boxplot(x = type2.Program, y = type2.ExecutionTime,showfliers = False, orient = "v", ax = ax[1])
plt.show()
```



Here, we could see UP350 has the highest execution time.

StartTime and Program: Count plot and box plot:

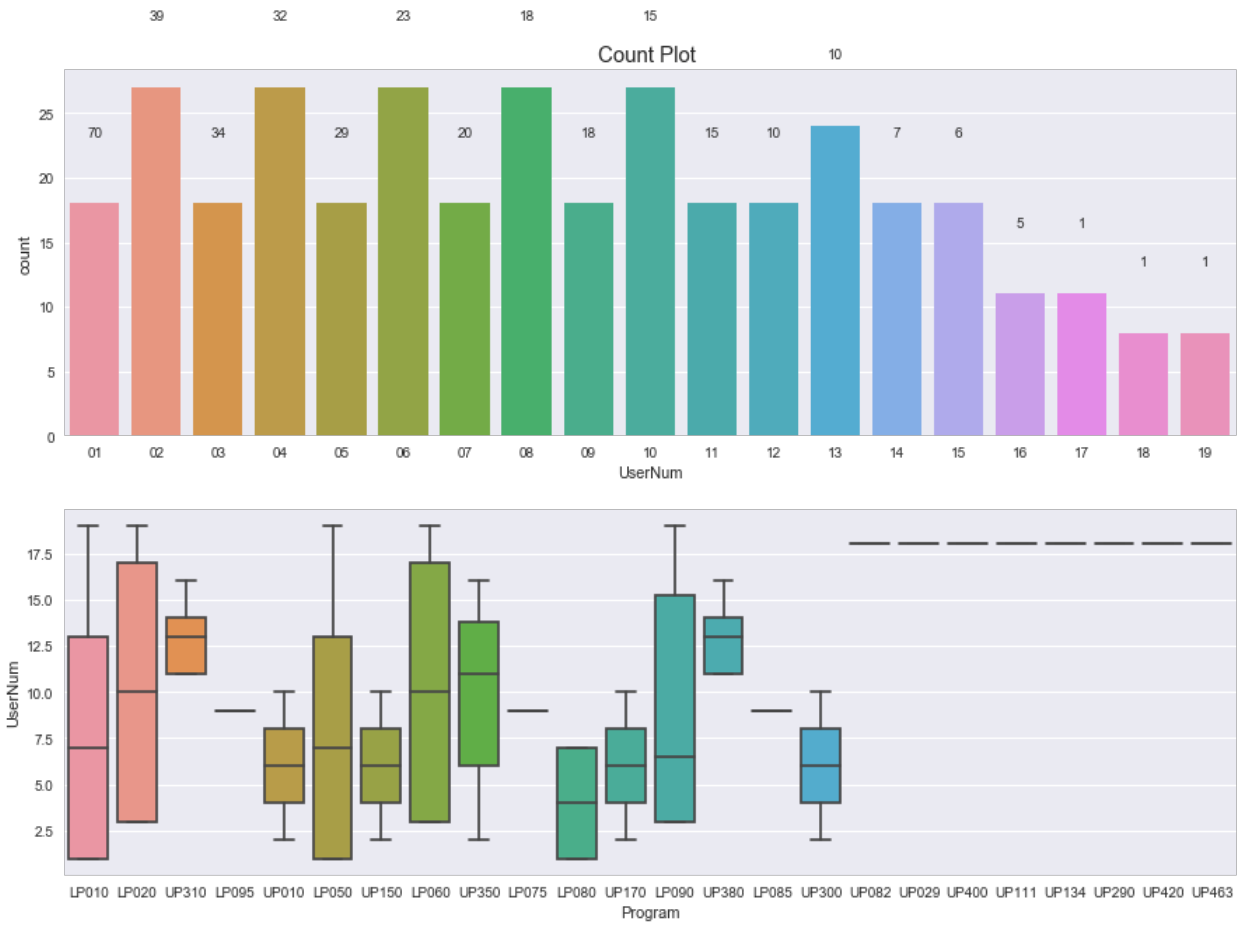
In [49]:
```
fig, ax = plt.subplots(2, 1, figsize = (15,10))
sns.countplot(type2.StartTime, ax = ax[0])
rectangles = ax[0].patches
ax[0].set_title("Count Plot ", fontsize = 15)
labels = type2.Program.value_counts().values
for rect, label in zip(rectangles, labels):
    height = rect.get_height()
    ax[0].text(rect.get_x() + rect.get_width()/2, height + 5, label, ha = "center", va = "bottom")
sns.boxplot(x = type2.Program, y = type2.StartTime,showfliers = False, orient = "v", ax = ax[1])
plt.show()
```



Here, we could infer that the data is very unevenly distributed for the programs.

User and Program: Count plot and box plot:

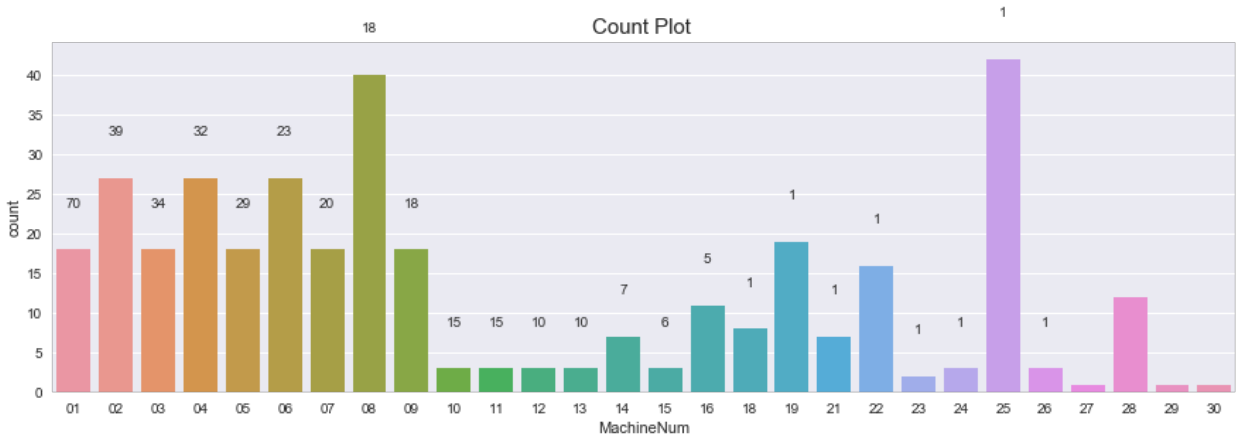
In [51]:
```
fig, ax = plt.subplots(2, 1, figsize = (15,10))
sns.countplot(type2.UserNum, ax = ax[0])
rectangles = ax[0].patches
ax[0].set_title("Count Plot ", fontsize = 15)
labels = type2.Program.value_counts().values
for rect, label in zip(rectangles, labels):
    height = rect.get_height()
    ax[0].text(rect.get_x() + rect.get_width()/2, height + 5, label, ha = "center", va = "bottom")
sns.boxplot(x = type2.Program, y = type2.UserNum.astype(np.float),showfliers = False, orient = "v", ax = ax[1])
plt.show()
```

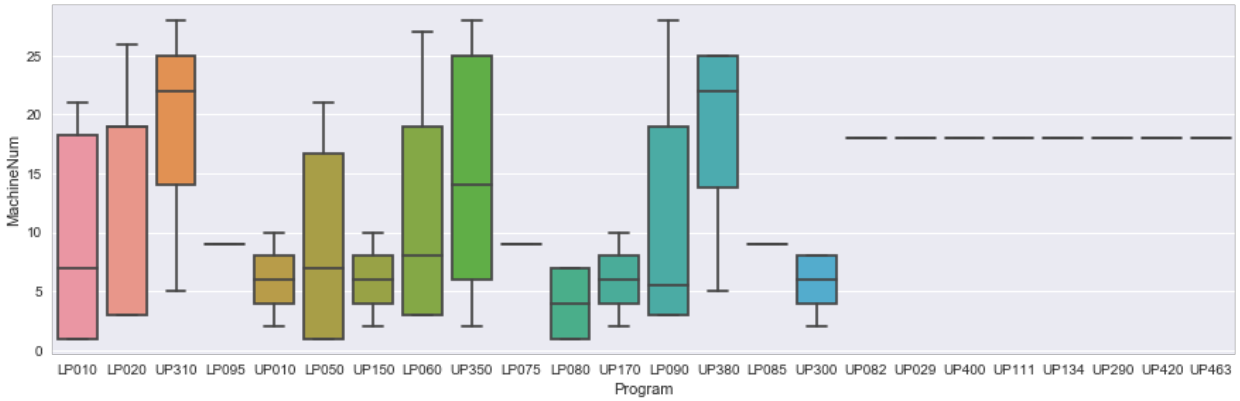


Here, we could see unevenly distributed data.

Machine and Program: Count plot and box plot:

In [52]:
```
fig, ax = plt.subplots(2, 1, figsize = (15,10))
sns.countplot(type2.MachineNum, ax = ax[0])
rectangles = ax[0].patches
ax[0].set_title("Count Plot ", fontsize = 15)
labels = type2.Program.value_counts().values
for rect, label in zip(rectangles, labels):
    height = rect.get_height()
    ax[0].text(rect.get_x() + rect.get_width()/2, height + 5, label, ha = "center", va = "bottom")
sns.boxplot(x = type2.Program, y = type2.MachineNum.astype(np.float),showfliers = False, orient = "v", ax = ax[1])
plt.show()
```

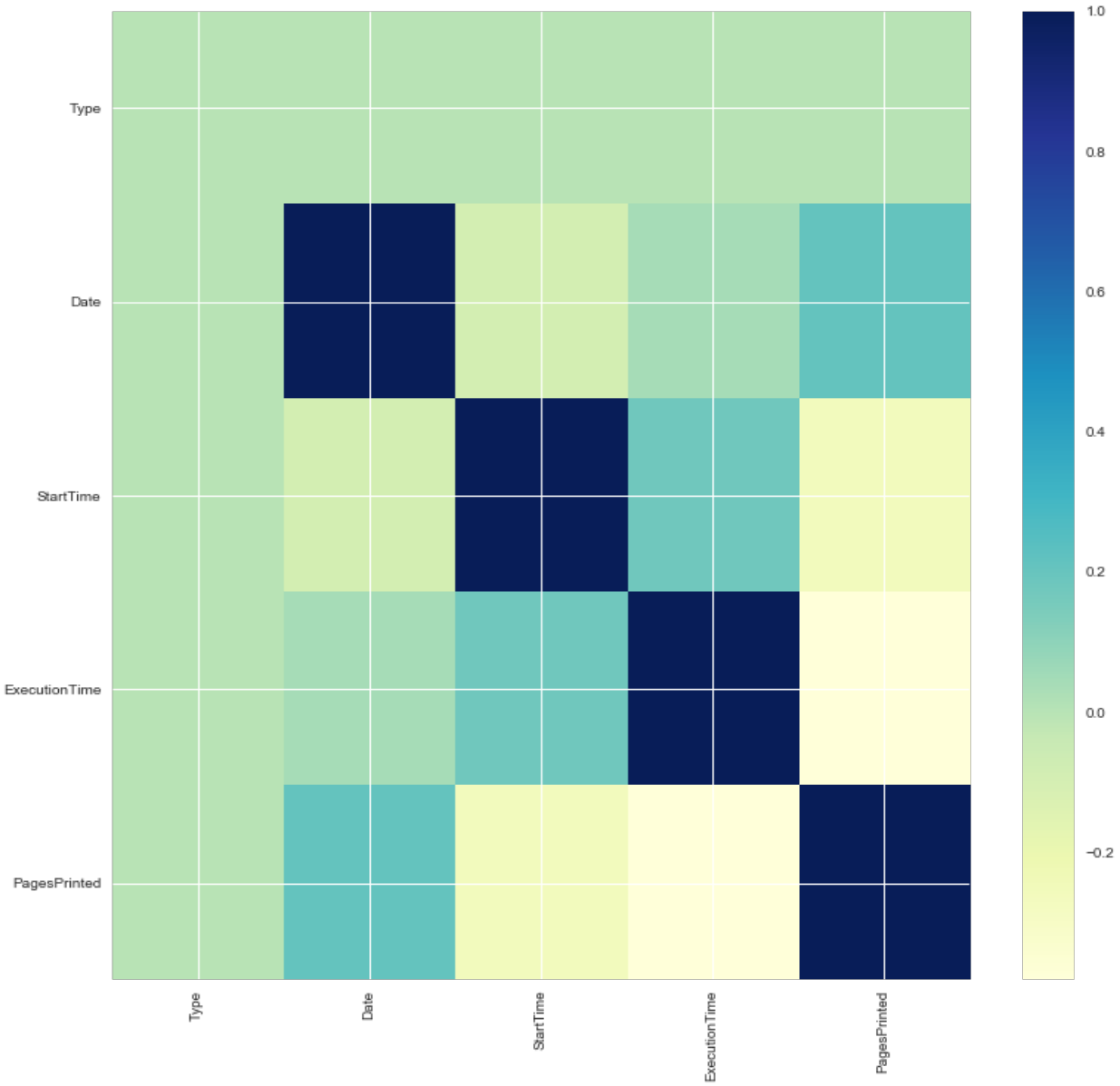




Here, we could see certain empty lines illustrate that no program is used by any of machines

Statistics: Correlation:

```
In [53]: corltn=type2.corr()
         corltn=corltn.fillna(0)
         plt.figure(figsize=(13, 13))
         plt.imshow(corltn, cmap='YlGnBu', interpolation='none', aspect='auto')
         plt.colorbar()
         plt.xticks(range(len(corltn)), corltn.columns, rotation='vertical')
         plt.yticks(range(len(corltn)), corltn.columns);
         plt.suptitle(' Correlations Heat Map for attributes', fontsize=16, fontweight='bold')
```



Converting ‘Program’ into numeric values:

In [56]:
```
def toNumeric(data,to):
    if type2[data].dtype == type(object):
        le = preprocessing.LabelEncoder()
        type2[to] = le.fit_transform(type2[data].astype(str))
toNumeric('Program','ProgramNum')
type2.head()
```

Out[56]:

	Type	User	Machine	Date	StartTime	Program	ExecutionTime	File	Status	Printer	PagesPrinted	UserNum	MachineNum	ProgramNum
0	2	U01	M01	90108	90201	LP010	340	F0059	R	PR1	10.0	01	01	0
1	2	U03	M03	90108	90201	LP020	340	F0059	R	PR1	10.0	03	03	1
2	2	U05	M05	90108	90201	UP310	340	F0010	RW	PR2	10.0	05	05	18
3	2	U07	M07	90108	90201	LP010	340	F0010	RW	PR2	10.0	07	07	0
4	2	U09	M09	90108	90201	LP095	340	F0010	RW	PR2	10.0	09	09	8

5.Building data model: Linear Regression:

In [57]:
```
X_train, x_test, Y_train, y_test = train_test_split(
type2['UserNum'], type2['ProgramNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
X_train = X_train[:, np.newaxis]
Y_train = Y_train[:, np.newaxis]
x_test = x_test[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
plt.scatter(X_train, Y_train, color='navy', s=30, marker='o', label="training points")
plt.plot(x_test, y_predict, color='teal', linewidth=2)
plt.legend(loc='best')
plt.show()
```

Performance evaluation:

In [58]:
```
r_data= type2[['UserNum','MachineNum','Date']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type2['ProgramNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

Out[58]:
```
48.143291498299931
```

In [59]:
```
r_data= type2[['UserNum','MachineNum','Date','ExecutionTime','StartTime']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type2['ProgramNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

Out[59]:
```
43.664818086878839
```

In [60]:
```
r_data= type2[['UserNum','MachineNum','Date','ExecutionTime','StartTime','PagesPrinted']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type2['ProgramNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

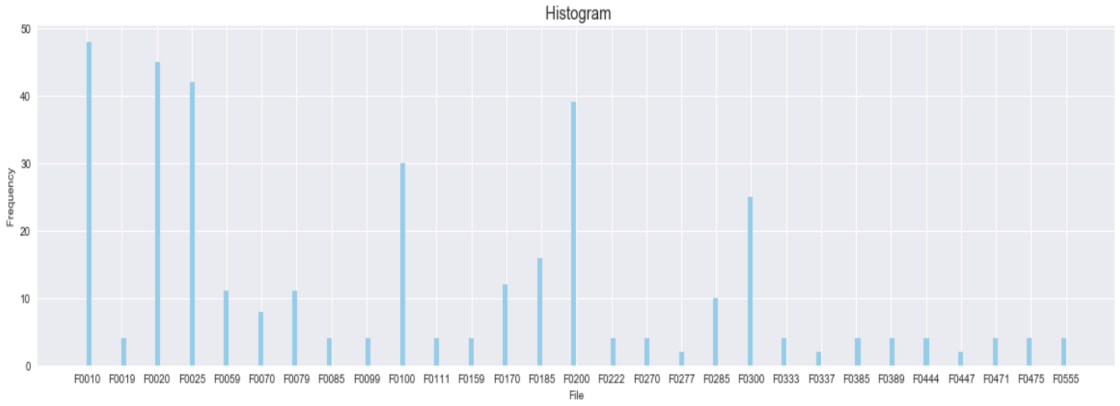
Out[60]:
```
43.998878668023977
```

Here, we obtained a considerably decent mean square error and we could use this model.

3. FILE ACCESS PATTERN:

File: Histogram:

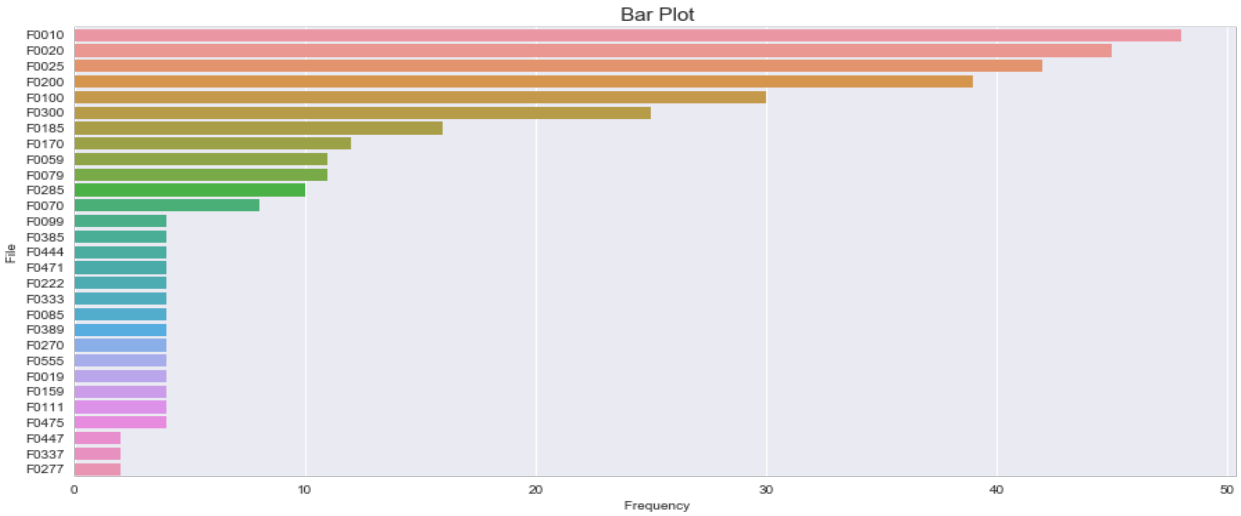
In [62]:
```
fig, ax = plt.subplots(1, 1, figsize = (20, 5))
ax.hist(type2.File, bins = 200, range = [min(type2.File), max(type2.File)], label = "price",color = "skyblue")
ax.set_title("\n \n  Histogram ", fontsize = 15)
ax.set_xlabel("File", fontsize = 10)
ax.set_ylabel(" Frequency ", fontsize = 10)
plt.show()
```



File: Bar plot and Scatter plot

In [63]:
```
file = type2["File"].value_counts()
print("Unique Files :", file.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(file.values, file.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type2['Status'].values
f2 = type2['File'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" File ", fontsize = 10)
ax[1].set_xlabel(" Status ", fontsize = 10)
ax[1].set_ylabel(" File ", fontsize = 10)
plt.show()
```

```
Unique Files : 29
```

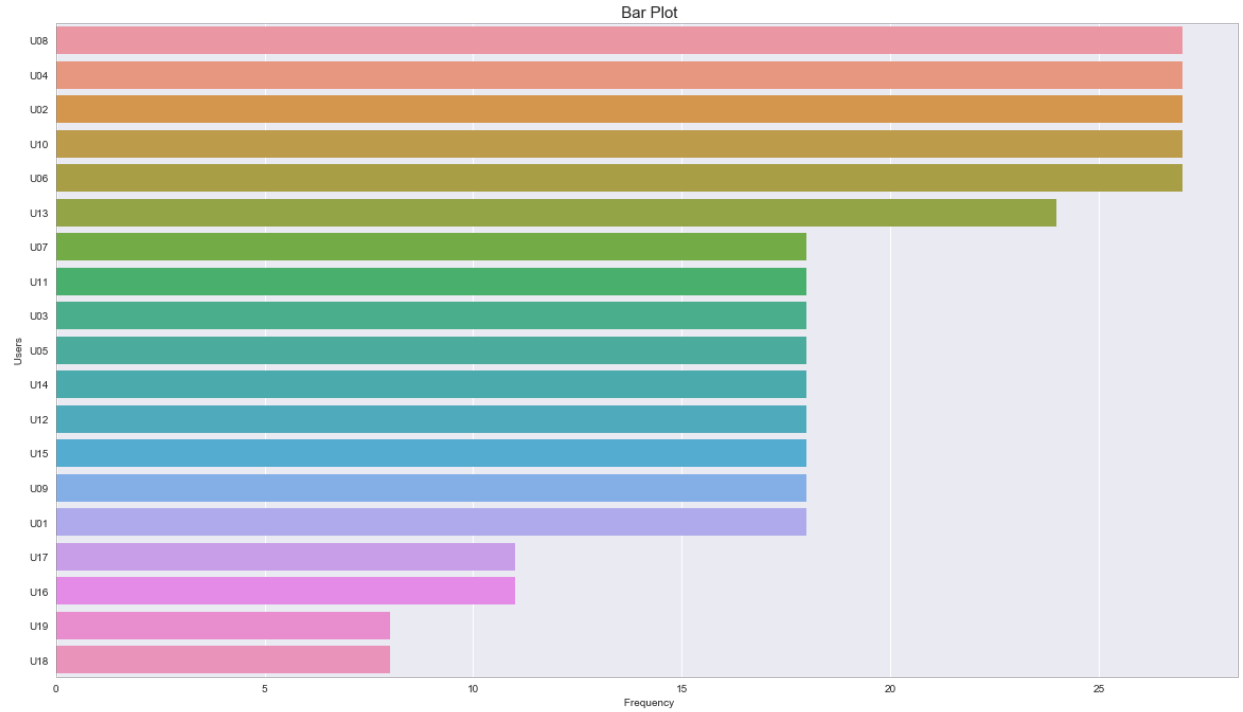


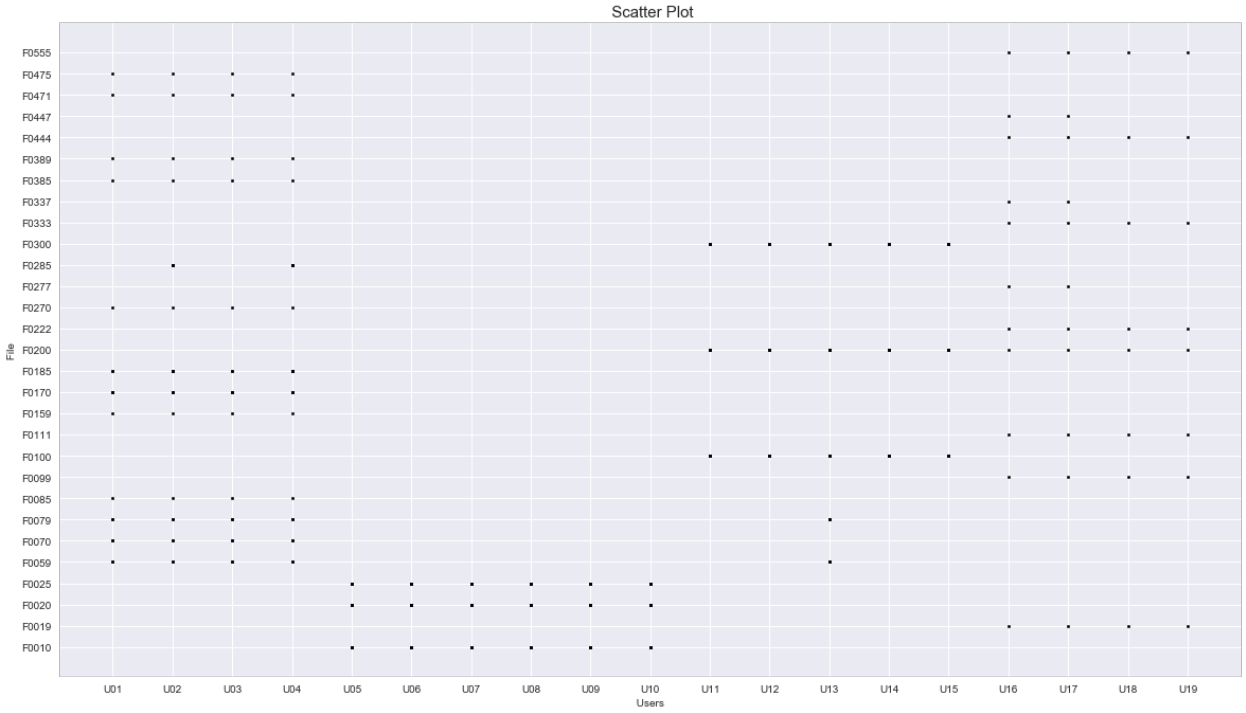
Here, we see that “RW” has more data distribution with the files accessed.

User and Files: Bar plot and Scatter plot

In [184]:
```
user = type2["User"].value_counts()
print("Unique Users :", user.size)
fig, ax = plt.subplots(2, 1, figsize = (20, 25))
sns.barplot(user.values, user.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" User", fontsize = 10)
f1 = type2['User'].values
f2 = type2['File'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" Users ", fontsize = 10)
ax[1].set_xlabel(" Users ", fontsize = 10)
ax[1].set_ylabel(" File ", fontsize = 10)
plt.show()
```

```
Unique Users : 19
```



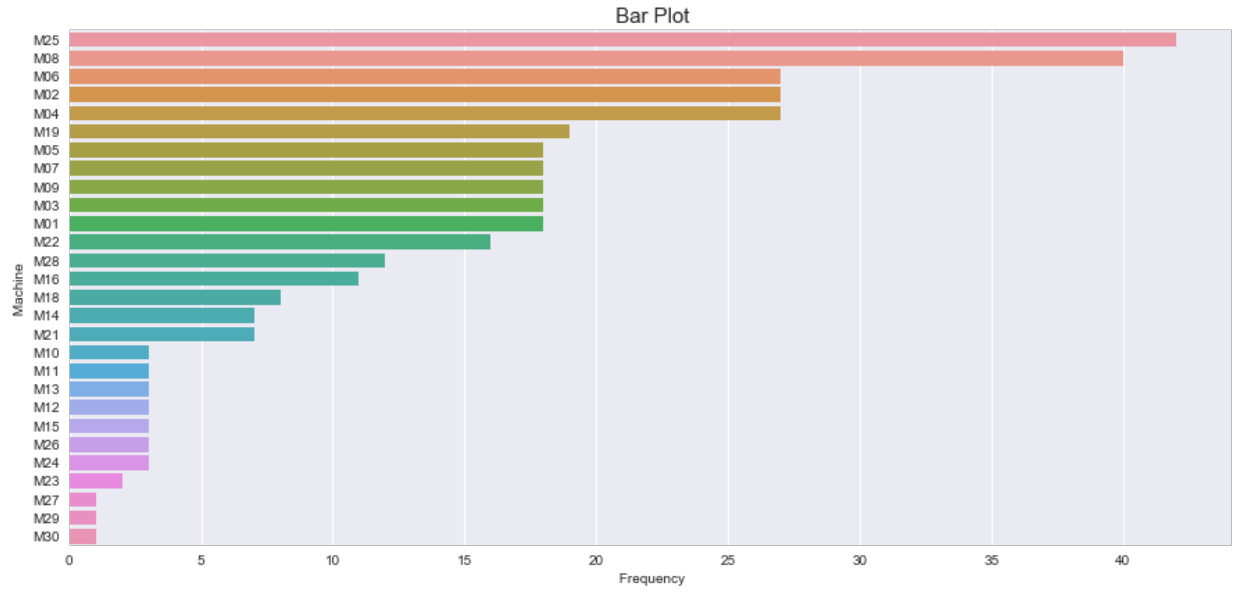


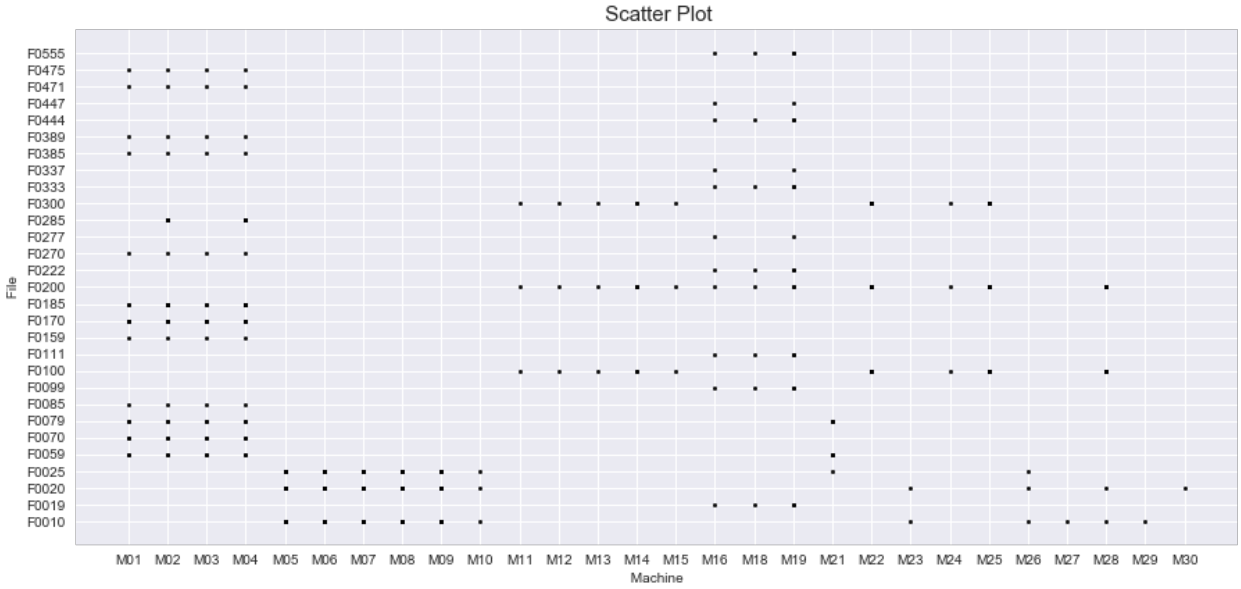
Here, we could find that data from U05 to U10 has a specific pattern in common with files.

Machine and file: Bar plot and Scatter plot.

In [185]:
```
machine = type2["Machine"].value_counts()
print("Unique Machines :", machine.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(machine.values, machine.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type2['Machine'].values
f2 = type2['File'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" Machine ", fontsize = 10)
ax[1].set_xlabel(" Machine ", fontsize = 10)
ax[1].set_ylabel(" File ", fontsize = 10)
plt.show()
```

```
Unique Machines : 28
```



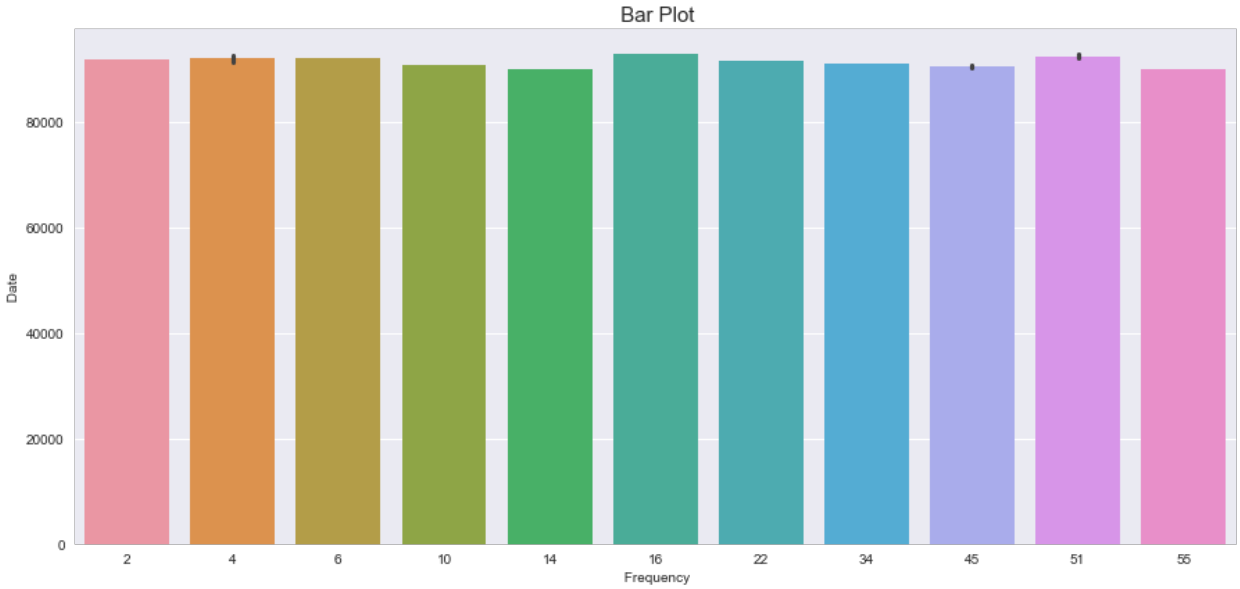


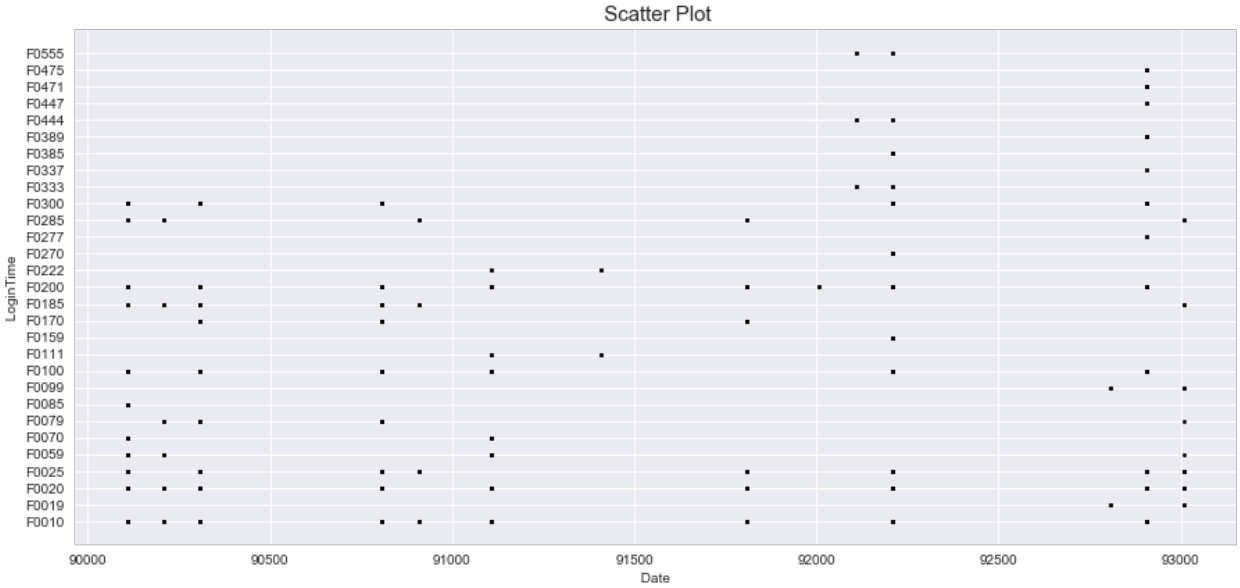
Here, we could find that there is a specific pattern for M05 to M10 with the files accessed.

Date and File: Bar plot and Scatter plot.

In [66]:
```
date = type2["Date"].value_counts()
print("Unique dates :", date.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(date.values, date.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type2['Date'].values
f2 = type2['File'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" Date ", fontsize = 10)
ax[1].set_xlabel(" Date ", fontsize = 10)
ax[1].set_ylabel(" LoginTime ", fontsize = 10)
plt.show()
```

```
Unique dates : 14
```



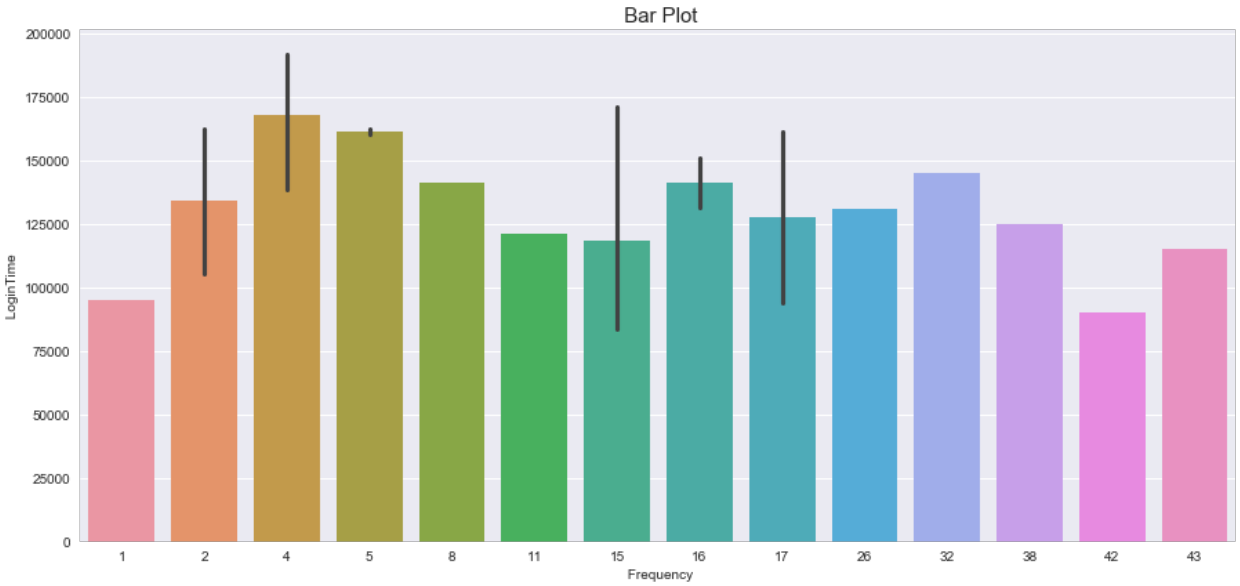


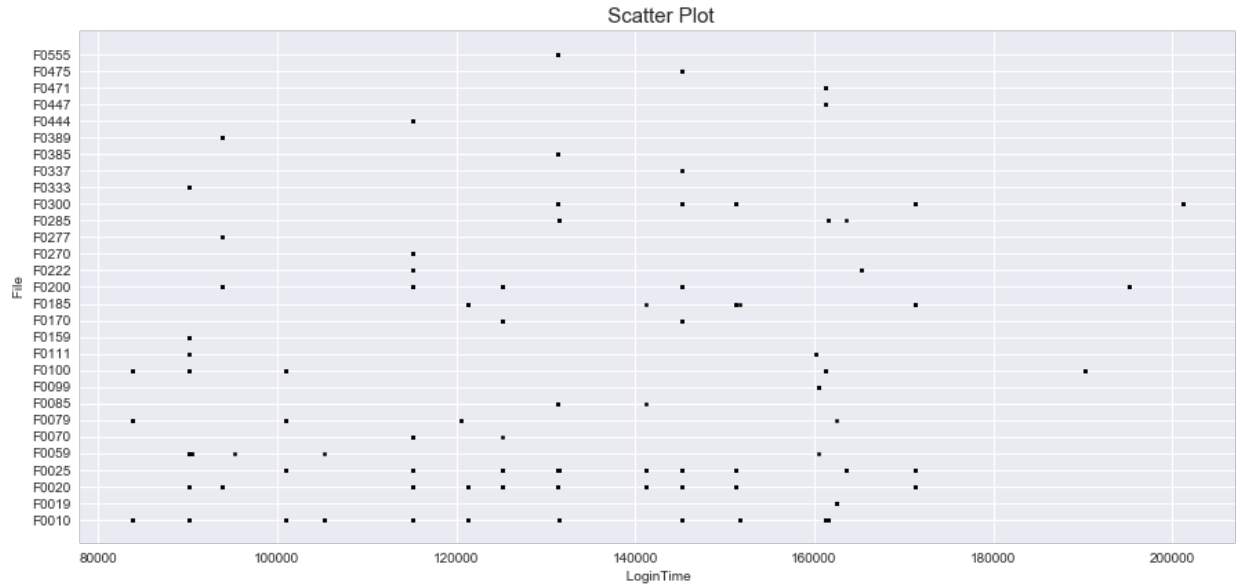
Here, we could notice no file is being accessed on 09500.

StartTime and File: Bar plot and Scatter plot.

In [67]:
```
starttime = type2["StartTime"].value_counts()
print("Unique starttime :", starttime.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(starttime.values, starttime.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type2['StartTime'].values
f2 = type2['File'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" LoginTime ", fontsize = 10)
ax[1].set_xlabel(" LoginTime ", fontsize = 10)
ax[1].set_ylabel(" File ", fontsize = 10)
plt.show()
```

```
Unique starttime : 28
```



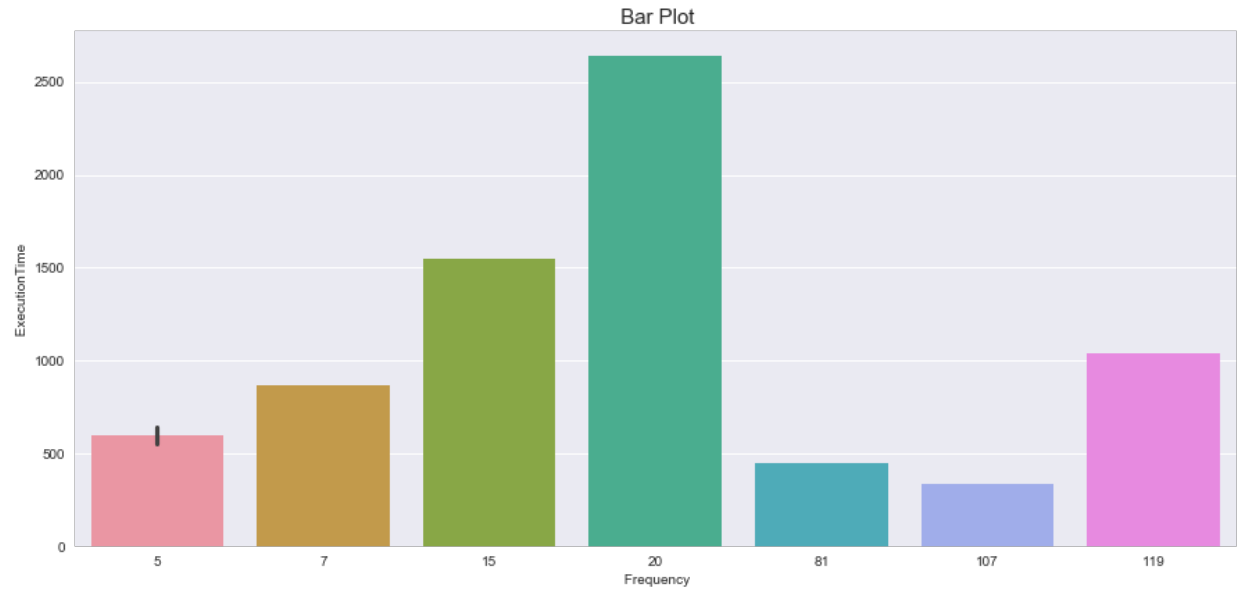


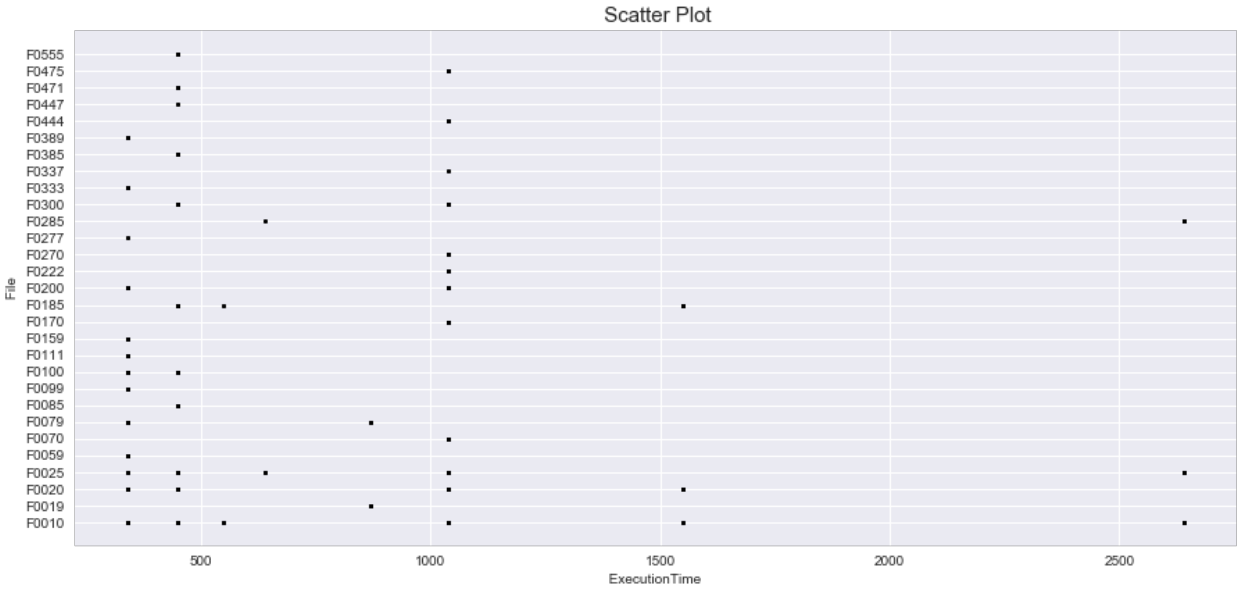
Here, we could notice data is randomly distributed and could not possibly conclude.

ExecutionTime and File: Bar plot and Scatter plot.

```
In [68]: executiontime = type2["ExecutionTime"].value_counts()
         print("Unique executiontime :", executiontime.size)
         fig, ax = plt.subplots(2, 1, figsize = (15, 15))
         sns.barplot(executiontime.values, executiontime.index,ax = ax[0])
         ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
         ax[0].set_xlabel(" Sale Price", fontsize = 10)
         f1 = type2['ExecutionTime'].values
         f2 = type2['File'].values
         X = np.array(list(zip(f1, f2)))
         ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
         ax[1].scatter(f1, f2, c='black', s=7)
         ax[0].set_xlabel(" Frequency", fontsize = 10)
         ax[0].set_ylabel(" ExecutionTime ", fontsize = 10)
         ax[1].set_xlabel(" ExecutionTime ", fontsize = 10)
         ax[1].set_ylabel(" File ", fontsize = 10)
         plt.show()
```

```
Unique executiontime : 8
```





Here, we could notice below 500 execution time has more data distributed with respect to files.

Conversion: Convert the type of 'File' to numeric.

In [69]:
```
def toNumeric(data,to):
    if type2[data].dtype == type(object):
        le = preprocessing.LabelEncoder()
        type2[to] = le.fit_transform(type2[data].astype(str))
toNumeric('File','FileNum')
type2.head()
```

Out[69]:

	Type	User	Machine	Date	StartTime	Program	ExecutionTime	File	Status	Printer	PagesPrinted	UserNum	MachineNum	ProgramNum	FileNum
0	2	U01	M01	90108	90201	LP010	340	F0059	R	PR1	10.0	01	01	0	4
1	2	U03	M03	90108	90201	LP020	340	F0059	R	PR1	10.0	03	03	1	4
2	2	U05	M05	90108	90201	UP310	340	F0010	RW	PR2	10.0	05	05	18	0
3	2	U07	M07	90108	90201	LP010	340	F0010	RW	PR2	10.0	07	07	0	0
4	2	U09	M09	90108	90201	LP095	340	F0010	RW	PR2	10.0	09	09	8	0

Building the model: Linear Regression:

In [70]:
```
X_train, x_test, Y_train, y_test = train_test_split(
type2['UserNum'], type2['FileNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
X_train = X_train[:, np.newaxis]
Y_train = Y_train[:, np.newaxis]
x_test = x_test[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
plt.scatter(X_train, Y_train, color='navy', s=30, marker='o', label="training points")
plt.plot(x_test, y_predict, color='teal', linewidth=2)
plt.legend(loc='best')
plt.show()
```

Performance evaluation:

In [74]:
```
r_data= type2[['UserNum','MachineNum','Date','PagesPrinted','ProgramNum']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type2['FileNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

Out[74]: 38.925095191029634

In [75]:
```
r_data= type2[['UserNum','MachineNum','Date','PagesPrinted','ProgramNum','StatusNum','PrinterNum']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type2['FileNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

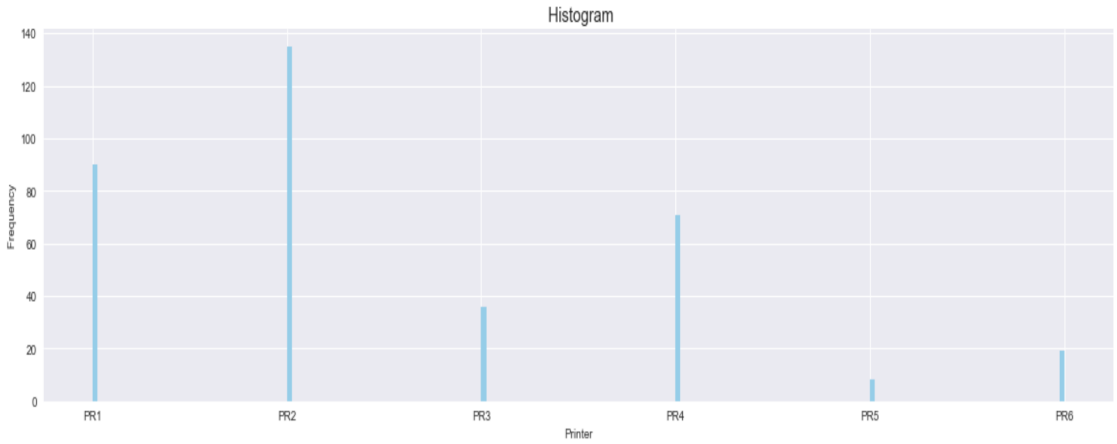
Out[75]: 34.98418164678408

Here, we obtained a decent mean square error and we could use this model for finding patterns.

PRINTER USAGE PATTERN:

Printer: Histogram:

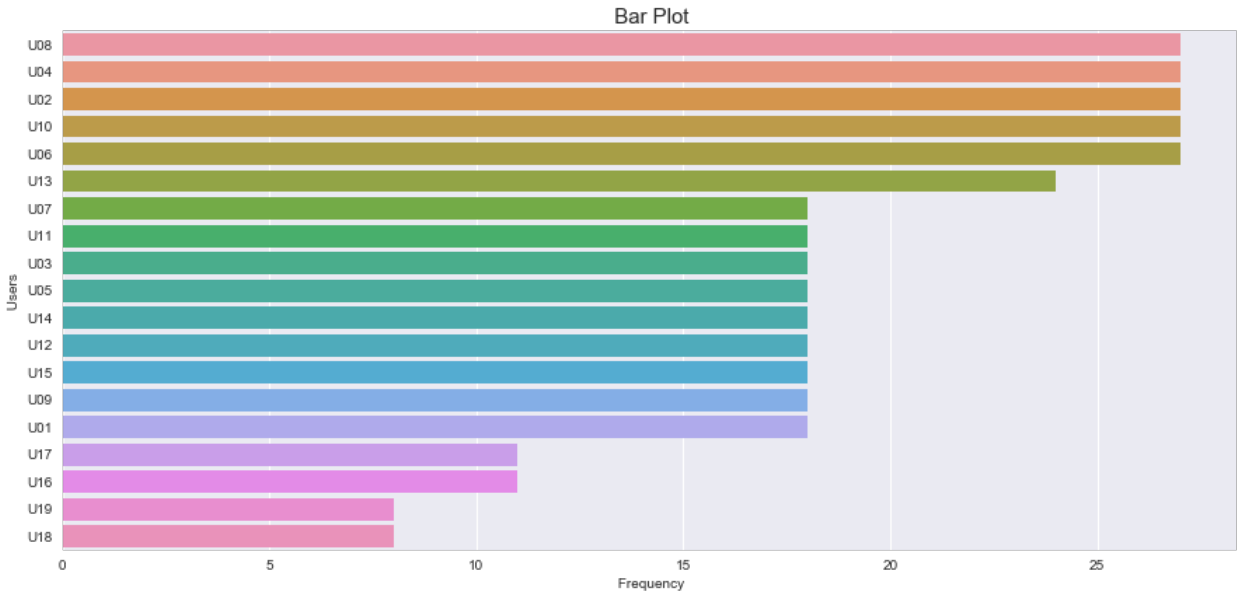
In [76]:
```
fig, ax = plt.subplots(1, 1, figsize = (20, 5))
ax.hist(type2.Printer, bins = 200, range = [min(type2.Printer), max(type2.Printer)], label = "price",color = "skyblue")
ax.set_title("\n \n  Histogram ", fontsize = 15)
ax.set_xlabel("Printer", fontsize = 10)
ax.set_ylabel(" Frequency ", fontsize = 10)
plt.show()
```

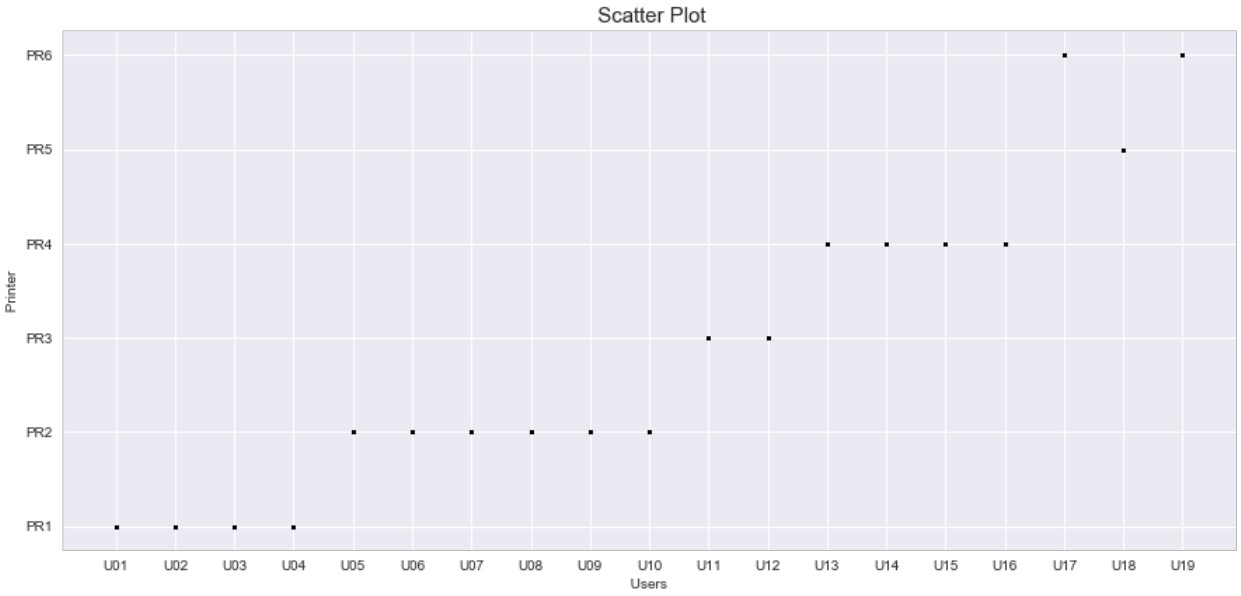


User: Bar plot and Scatter plot.

In [78]:
```
user = type2["User"].value_counts()
print("Unique users :", user.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(user.values, user.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
f1 = type2['User'].values
f2 = type2['Printer'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" Users ", fontsize = 10)
ax[1].set_xlabel(" Users ", fontsize = 10)
ax[1].set_ylabel(" Printer ", fontsize = 10)
plt.show()
```

```
Unique users : 19
```



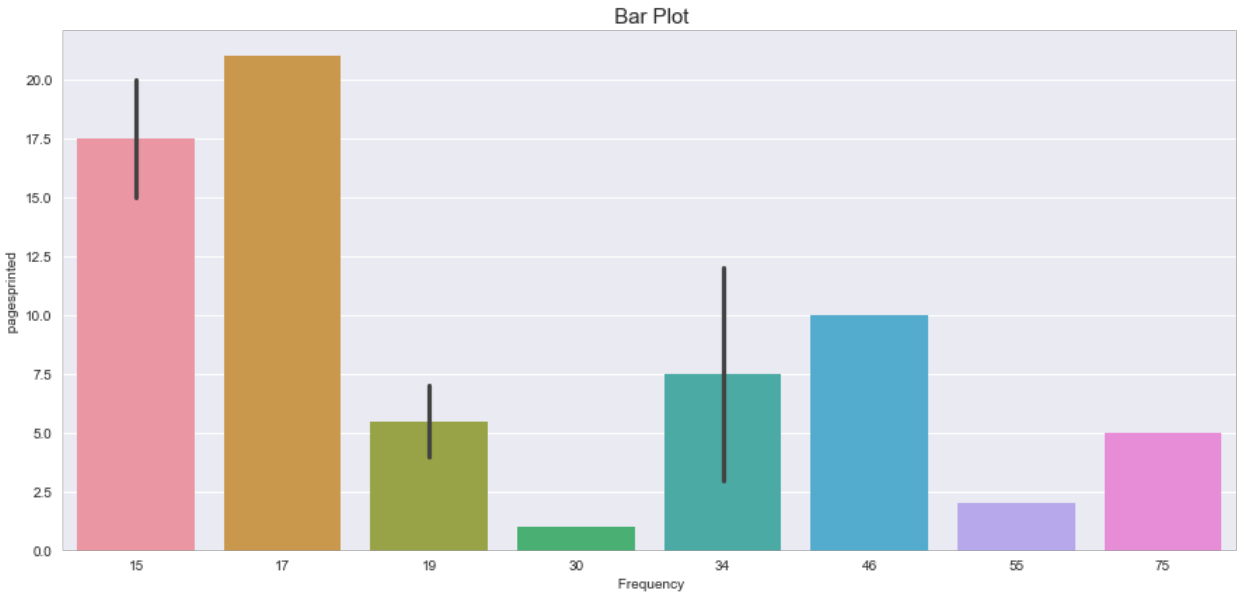


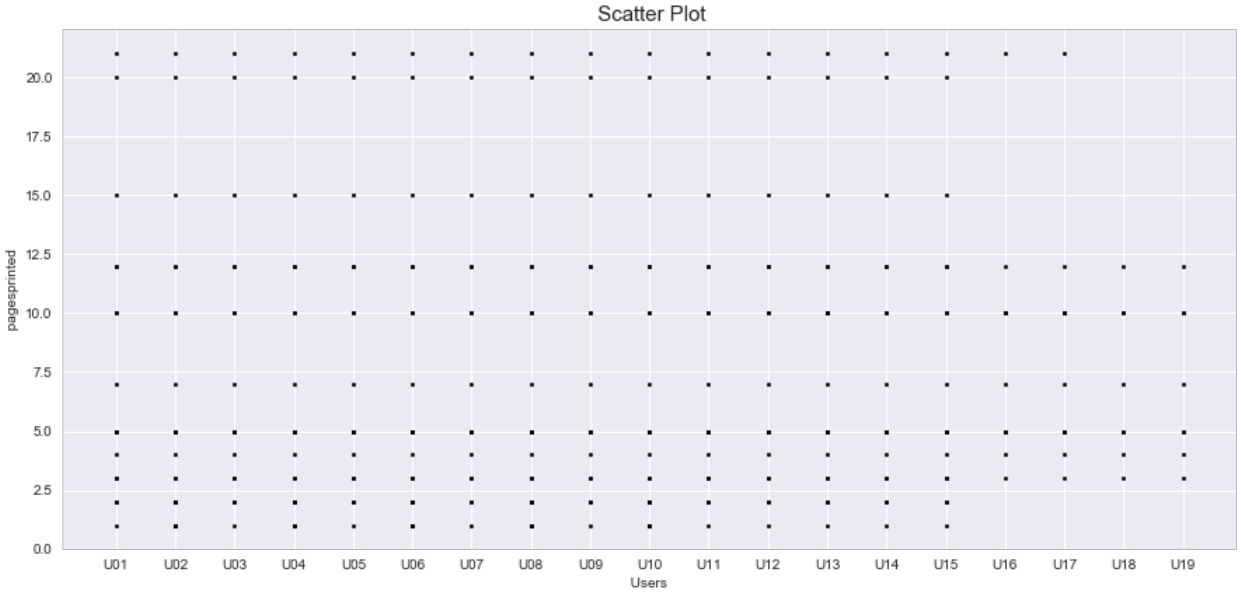
Here, we could notice specific users are using a single printer and there could be possibly grouped into clusters.

PagesPrinted: Bar plot and Scatter plot.

In [79]:
```
pagesprinted = type2["PagesPrinted"].value_counts()
print("Unique pagesprinted :", pagesprinted.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(pagesprinted.values, pagesprinted.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type2['User'].values
f2 = type2['PagesPrinted'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" pagesprinted ", fontsize = 10)
ax[1].set_xlabel(" Users ", fontsize = 10)
ax[1].set_ylabel(" pagesprinted ", fontsize = 10)
plt.show()
```

```
Unique pagesprinted : 11
```

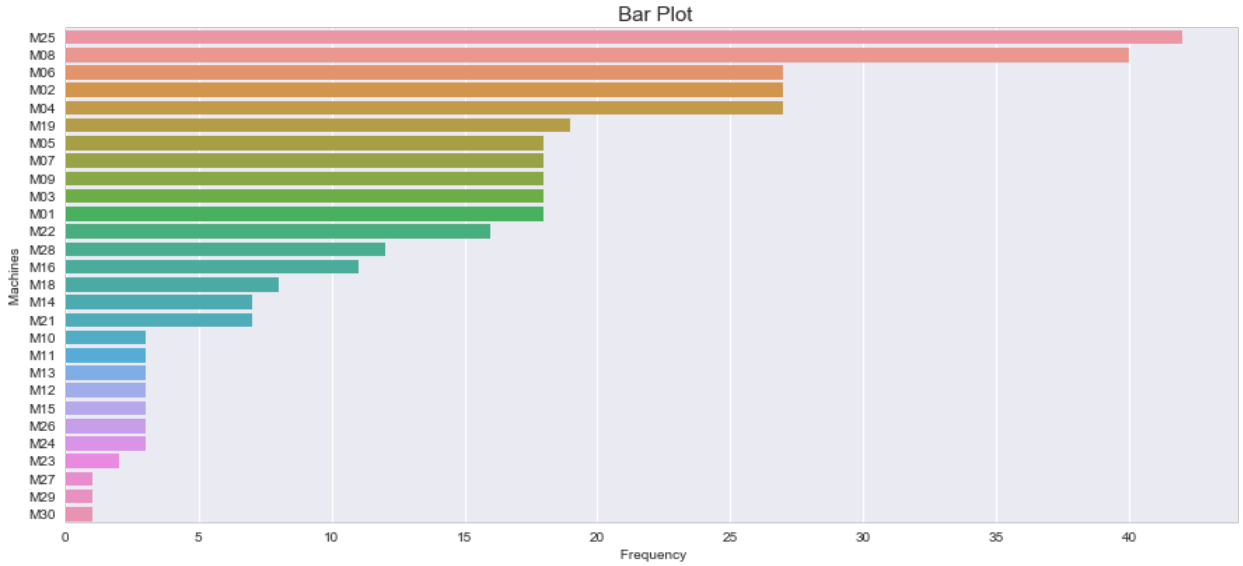


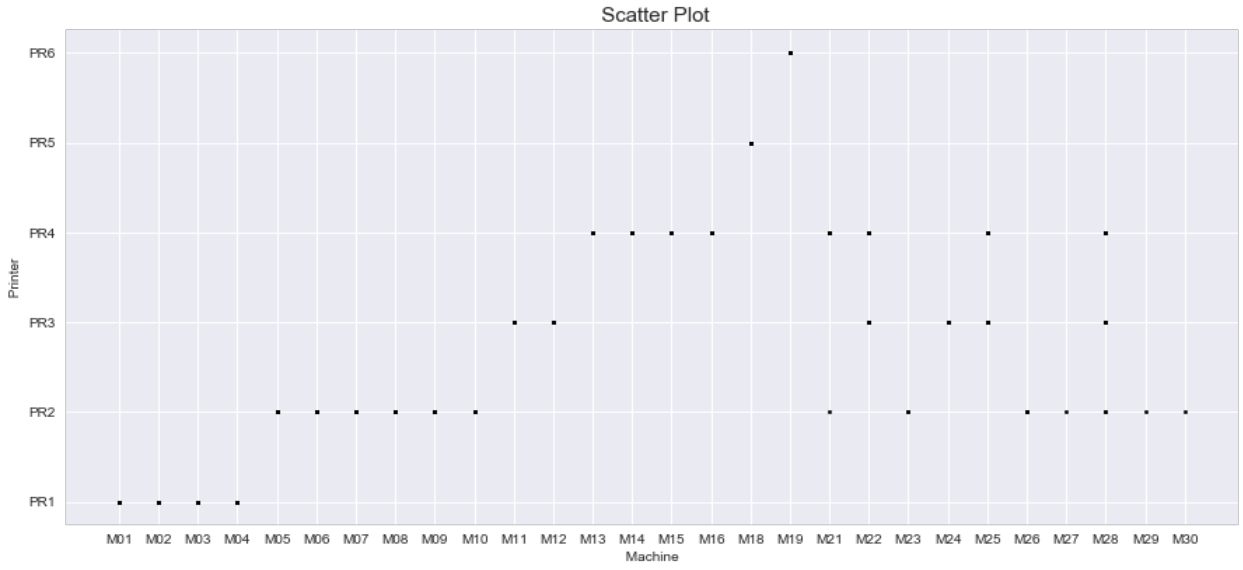


Here, we could find almost all the users have a similar pattern for checking number of pages printed.

Machine and Printer: Bar plot and Scatter plot.

```
In [80]: machine = type2["Machine"].value_counts()
         print("Unique Machines :", machine.size)
         fig, ax = plt.subplots(2, 1, figsize = (15, 15))
         sns.barplot(machine.values, machine.index,ax = ax[0])
         ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
         ax[0].set_xlabel(" Sale Price", fontsize = 10)
         f1 = type2['Machine'].values
         f2 = type2['Printer'].values
         X = np.array(list(zip(f1, f2)))
         ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
         ax[1].scatter(f1, f2, c='black', s=7)
         ax[0].set_xlabel(" Frequency", fontsize = 10)
         ax[0].set_ylabel(" Machines ", fontsize = 10)
         ax[1].set_xlabel(" Machine ", fontsize = 10)
         ax[1].set_ylabel(" Printer ", fontsize = 10)
         plt.show()

         Unique Machines : 28
```

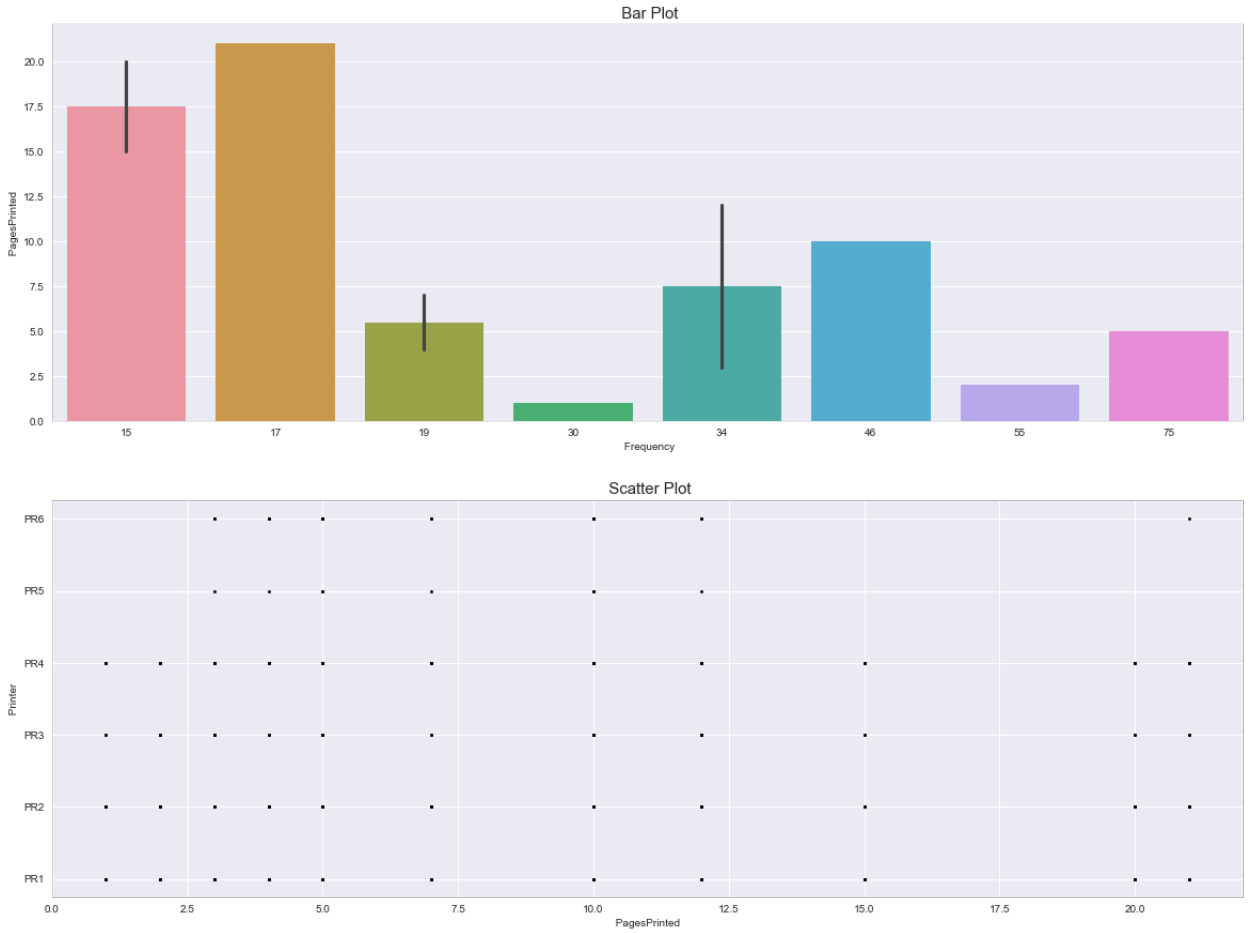




Here, we could see a random distribution of the machines and printers and possibly could not conclude for better results.

PagesPrinted: Bar plot and Scatter plot.

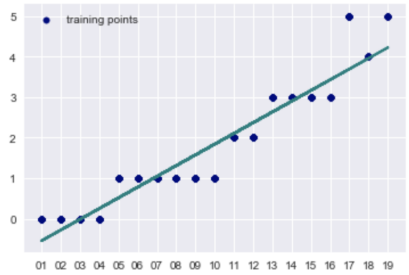
```
In [81]: pages = type2["PagesPrinted"].value_counts()
         print("Unique Pages :", pages.size)
         fig, ax = plt.subplots(2, 1, figsize = (20, 15))
         sns.barplot(pages.values, pages.index,ax = ax[0])
         ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
         f1 = type2['PagesPrinted'].values
         f2 = type2['Printer'].values
         X = np.array(list(zip(f1, f2)))
         ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
         ax[1].scatter(f1, f2, c='black', s=7)
         ax[0].set_xlabel(" Frequency", fontsize = 10)
         ax[0].set_ylabel(" PagesPrinted ", fontsize = 10)
         ax[1].set_xlabel(" PagesPrinted ", fontsize = 10)
         ax[1].set_ylabel(" Printer ", fontsize = 10)
         plt.show()

         Unique Pages : 11
```



Here, we find no printer has printed 17.5 pages.

Building the model: Linear Regression:

In [83]:
```
X_train, x_test, Y_train, y_test = train_test_split(
type2['UserNum'], type2['PrinterNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
X_train = X_train[:, np.newaxis]
Y_train = Y_train[:, np.newaxis]
x_test = x_test[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
plt.scatter(X_train, Y_train, color='navy', s=30, marker='o', label="training points")
plt.plot(x_test, y_predict, color='teal', linewidth=2)
plt.legend(loc='best')
plt.show()
```



In [84]:
```
r_data= type2[['UserNum','MachineNum','Date']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type2['PrinterNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

Out[84]: 0.1718518358982829

In [85]:
```
r_data= type2[['UserNum','MachineNum','Date','PagesPrinted','ProgramNum','StatusNum','FileNum']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type2['PrinterNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

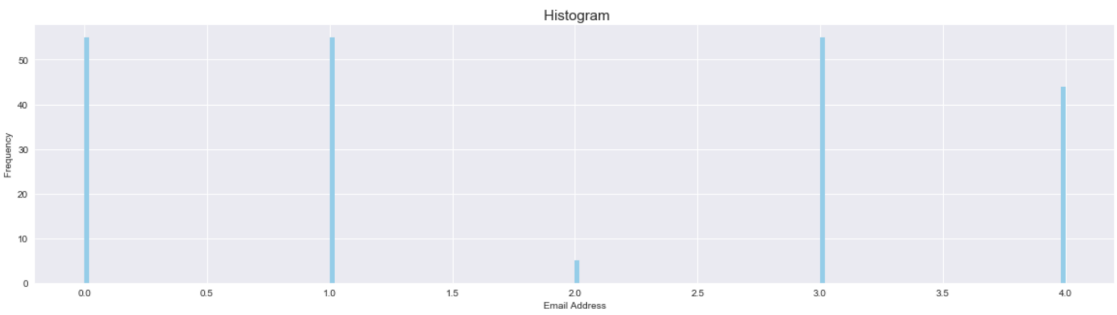
Out[85]: 0.16372843086445305

Here, we achieved a good mean square error and we would highly recommend this model.

EMAIL PATTERN:

EMail: Histogram

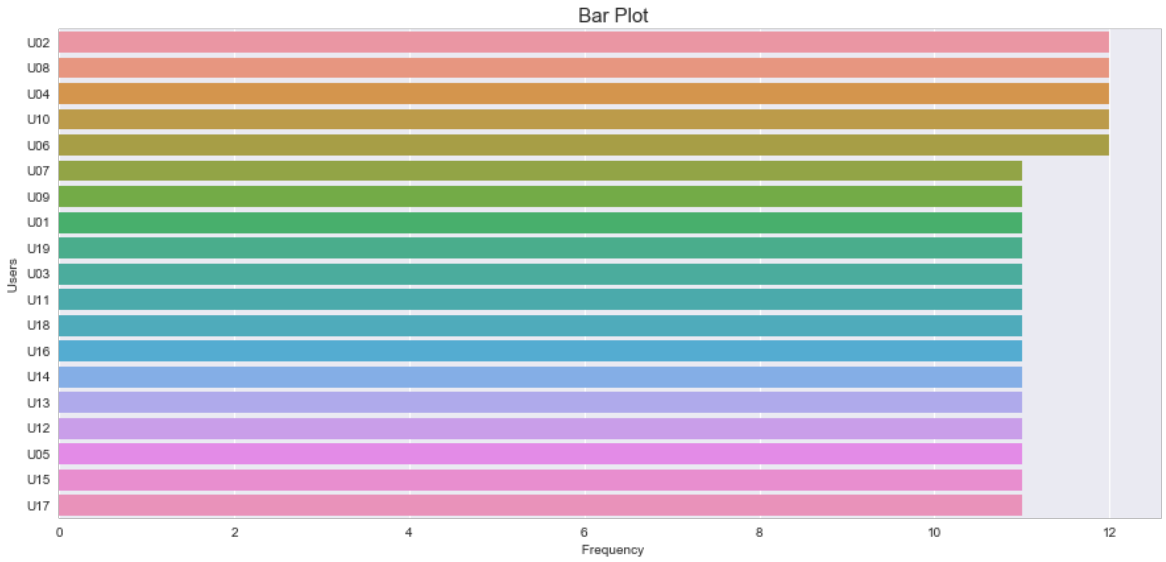
In [90]:
```
fig, ax = plt.subplots(1, 1, figsize = (20, 5))
ax.hist(type3.EmailAddressNum, bins = 200, range = [min(type3.EmailAddressNum), max(type3.EmailAddressNum)], label = "p
ax.set_title("\n \n  Histogram ", fontsize = 15)
ax.set_xlabel("Email Address", fontsize = 10)
ax.set_ylabel(" Frequency ", fontsize = 10)
plt.show()
```

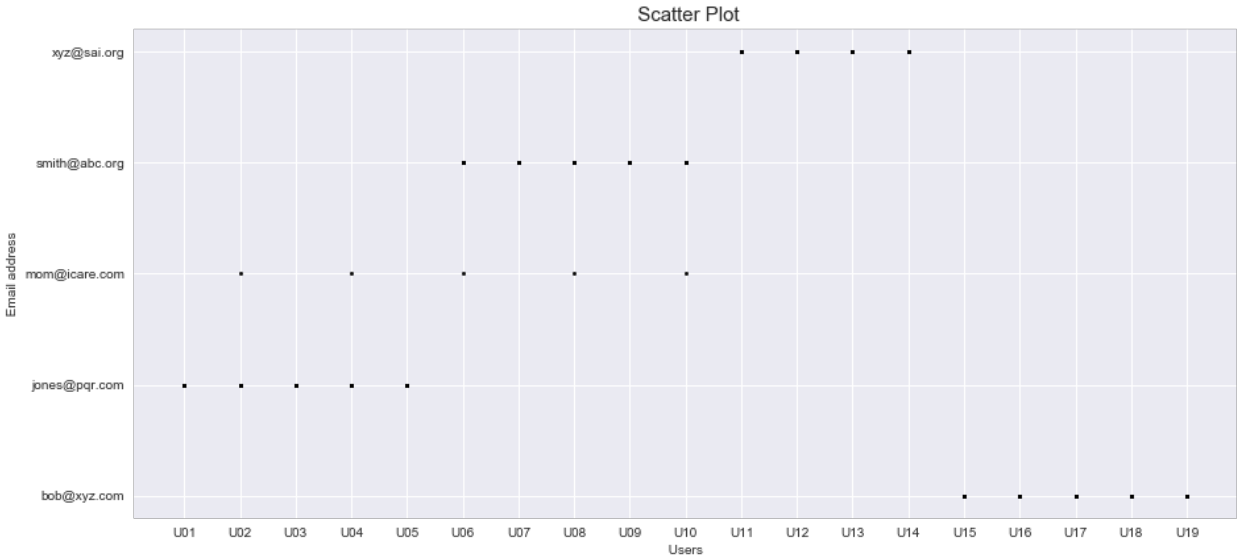


User:

```
In [91]: user = type3["User"].value_counts()
         print("Unique users :", user.size)
         fig, ax = plt.subplots(2, 1, figsize = (15, 15))
         sns.barplot(user.values, user.index,ax = ax[0])
         ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
         ax[0].set_xlabel(" Sale Price", fontsize = 10)
         f1 = type3['User'].values
         f2 = type3['EmailAddress'].values
         X = np.array(list(zip(f1, f2)))
         ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
         ax[1].scatter(f1, f2, c='black', s=7)
         ax[0].set_xlabel(" Frequency", fontsize = 10)
         ax[0].set_ylabel(" Users ", fontsize = 10)
         ax[1].set_xlabel(" Users ", fontsize = 10)
         ax[1].set_ylabel(" Email address ", fontsize = 10)
         plt.show()
```

```
Unique users : 19
```



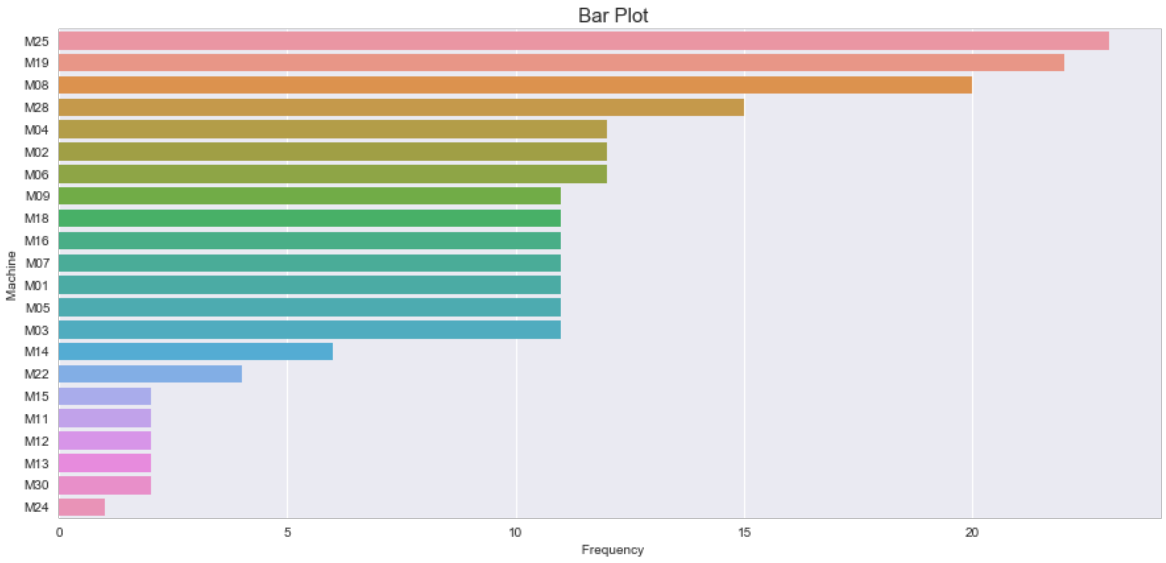


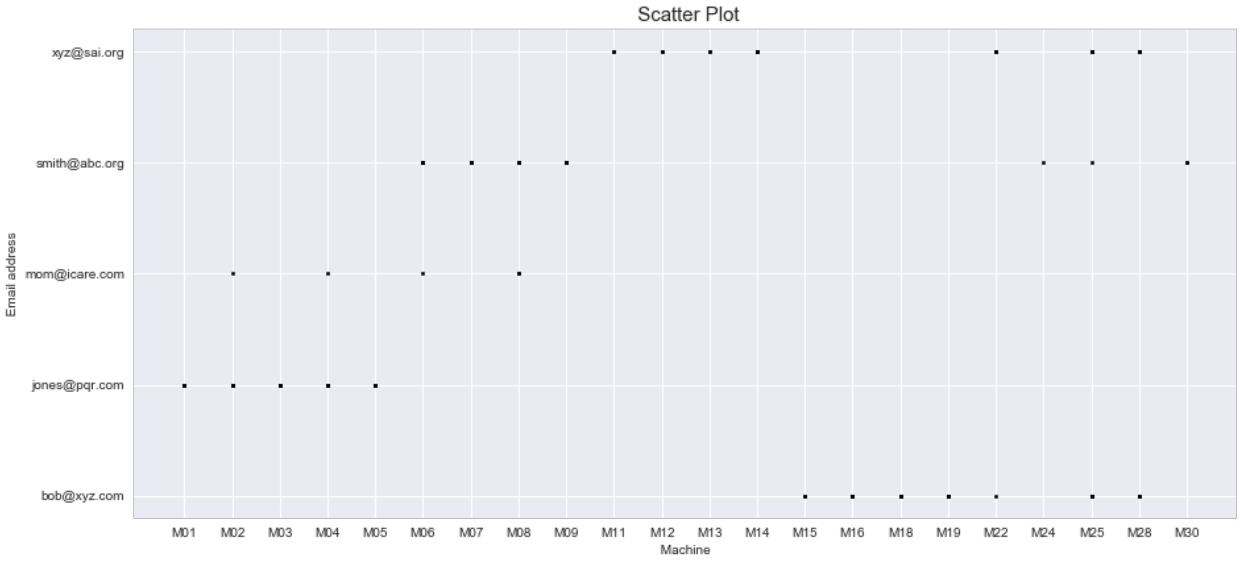
Here, U02, U08, U04, U10, U06 has highest and same email frequency counts.

Machine: Bar plot and Scatter plot.

In [92]:
```
machine = type3["Machine"].value_counts()
print("Unique Machines :", machine.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(machine.values, machine.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type3['Machine'].values
f2 = type3['EmailAddress'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" Machine ", fontsize = 10)
ax[1].set_xlabel(" Machine ", fontsize = 10)
ax[1].set_ylabel(" Email address ", fontsize = 10)
plt.show()
```

```
Unique Machines : 22
```



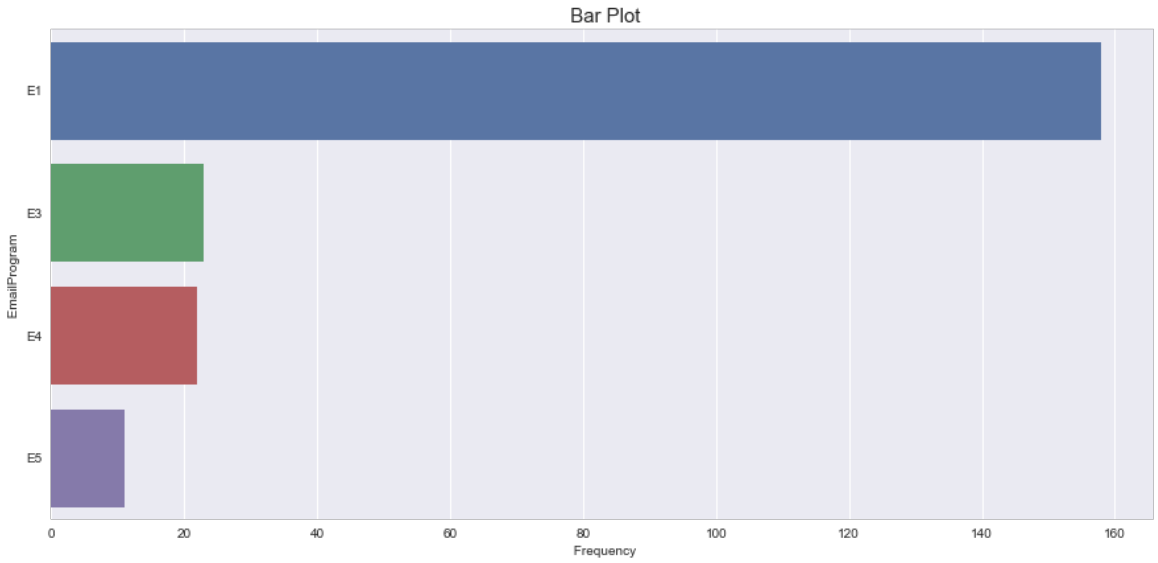


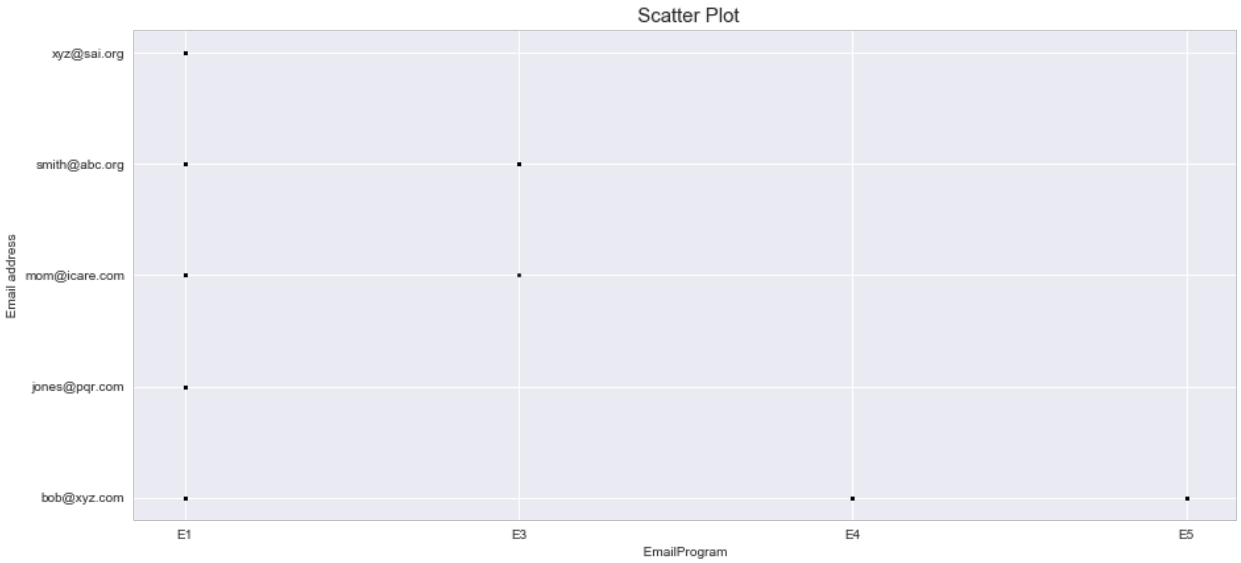
Here, we could see M25 has the highest counts of the emails.

EmailProgram: Bar plot and Scatter plot.

In [93]:
```
emailProgram = type3["EmailProgram"].value_counts()
print("Unique emailProgram :", emailProgram.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(emailProgram.values, emailProgram.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type3['EmailProgram'].values
f2 = type3['EmailAddress'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" EmailProgram ", fontsize = 10)
ax[1].set_xlabel(" EmailProgram ", fontsize = 10)
ax[1].set_ylabel(" Email address ", fontsize = 10)
plt.show()
```

```
Unique emailProgram : 4
```



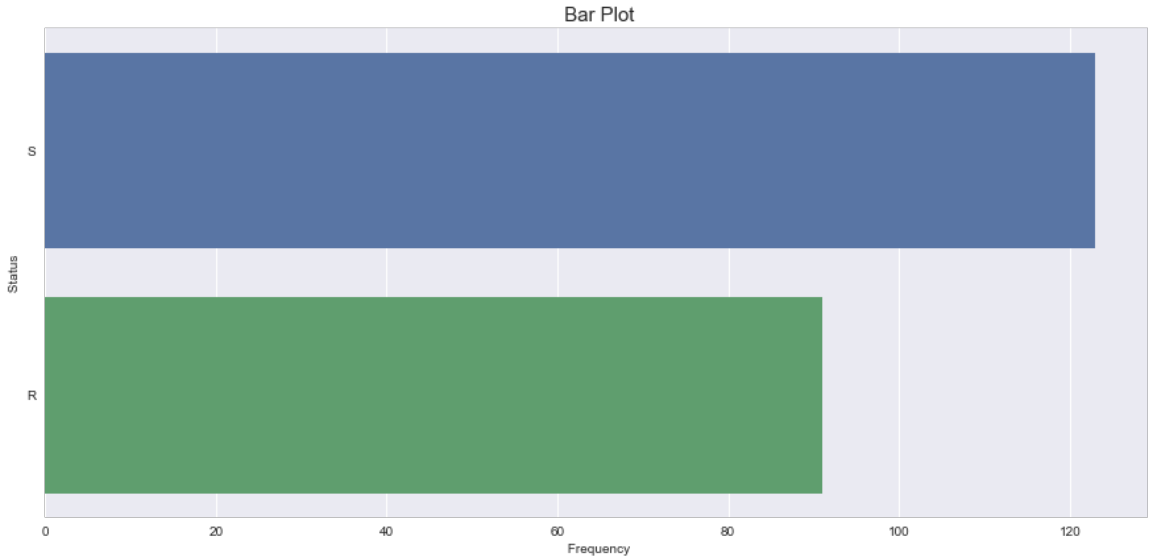


Here, we infer E1 has linked up with all the unique emails.

Status: Bar plot and Scatter plot.

In [94]:
```
status = type3["Status"].value_counts()
print("Unique status :", status.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(status.values, status.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type3['Status'].values
f2 = type3['EmailAddress'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" Status ", fontsize = 10)
ax[1].set_xlabel(" Status ", fontsize = 10)
ax[1].set_ylabel(" Email address ", fontsize = 10)
plt.show()
```

```
Unique status : 2
```

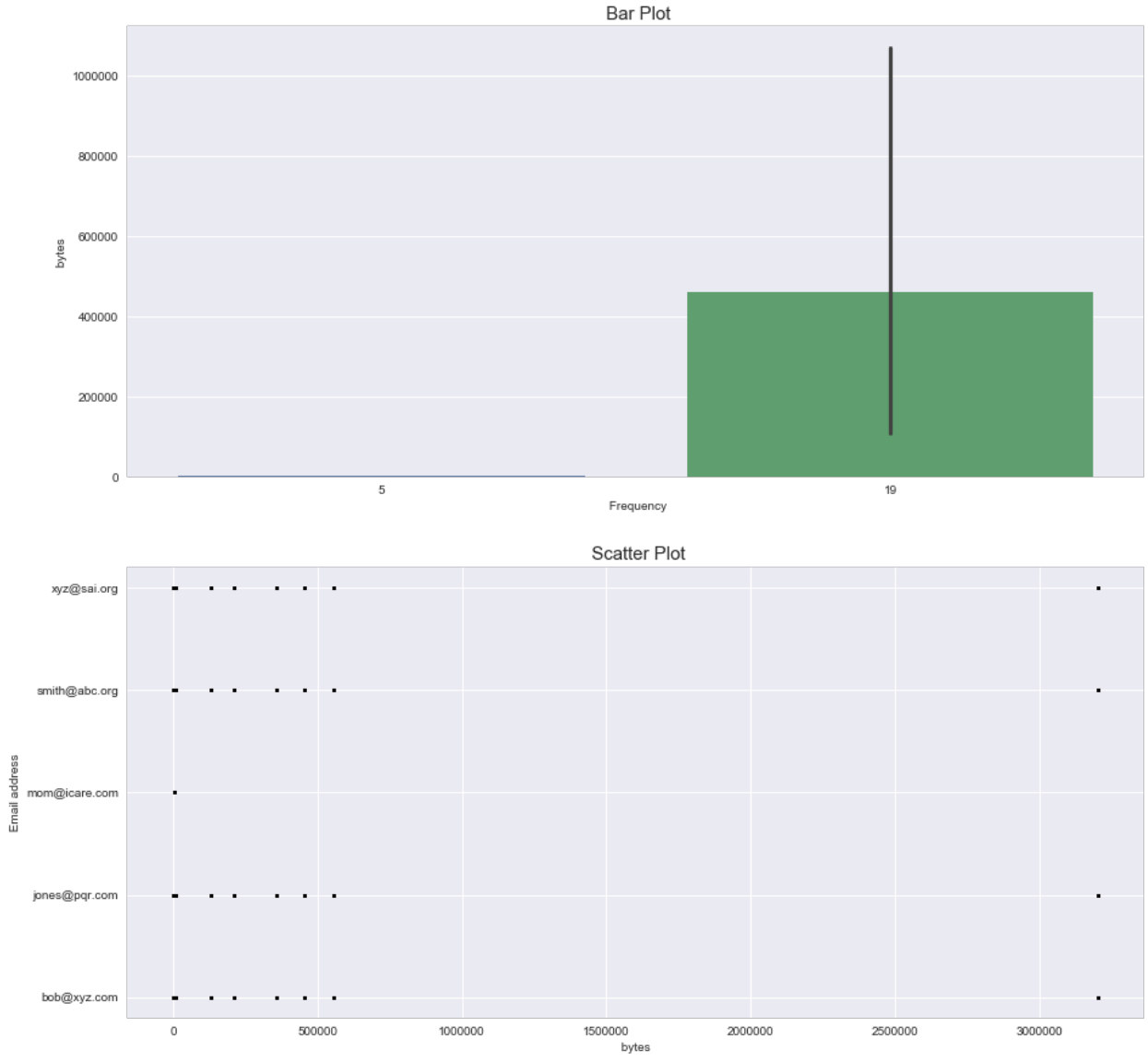


Here, we notice status “S” has more connectivity with the emails.

Bytes: Bar plot and Scatter plot.

In [187]:
```
bytes3 = type3["Bytes"].value_counts()
print("Unique status :", bytes3.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(bytes3.values, bytes3.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
f1 = type3['Bytes'].values
f2 = type3['EmailAddress'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" bytes ", fontsize = 10)
ax[1].set_xlabel(" bytes ", fontsize = 10)
ax[1].set_ylabel(" Email address ", fontsize = 10)
plt.show()
```

```
Unique status : 12
```

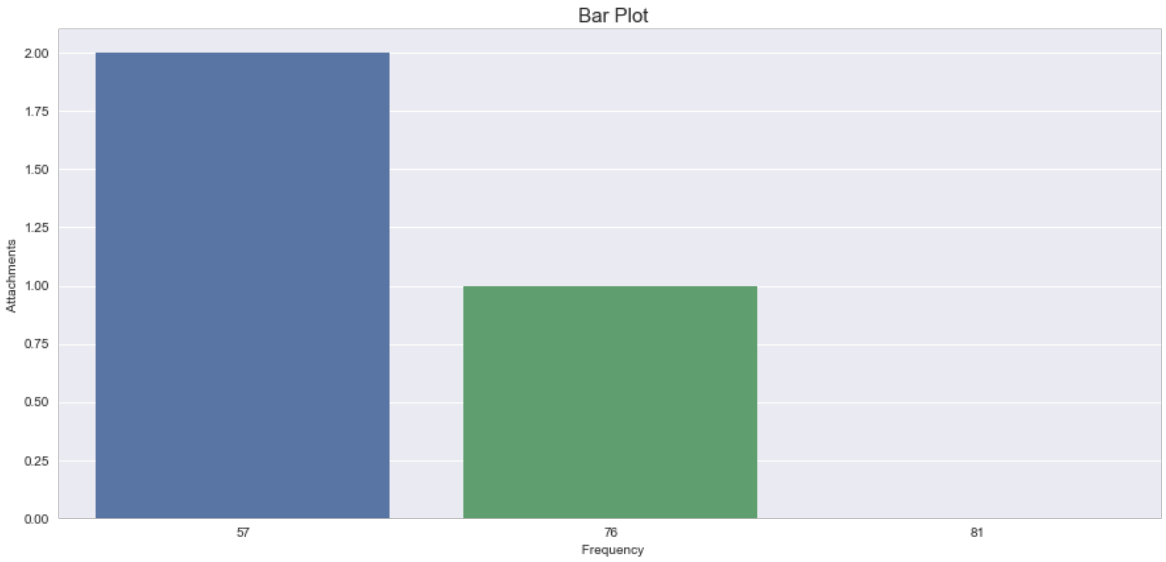


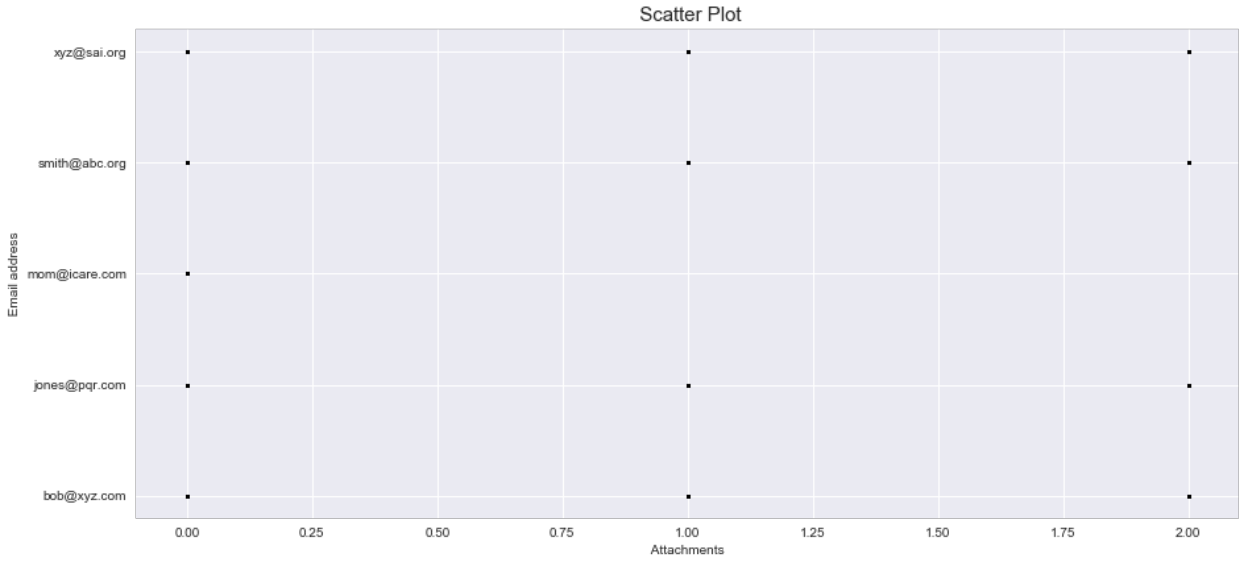
Here, we can notice from 0 to 500000 bytes there exists a similar behaviour among all emails.

Attachments: Bar plot and Scatter plot.

In [96]:
```
attachments = type3["Attachments"].value_counts()
print("Unique attachments :", attachments.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 15))
sns.barplot(attachments.values, attachments.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type3['Attachments'].values
f2 = type3['EmailAddress'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" Frequency", fontsize = 10)
ax[0].set_ylabel(" Attachments ", fontsize = 10)
ax[1].set_xlabel(" Attachments ", fontsize = 10)
ax[1].set_ylabel(" Email address ", fontsize = 10)
plt.show()
```

```
Unique attachments : 3
```

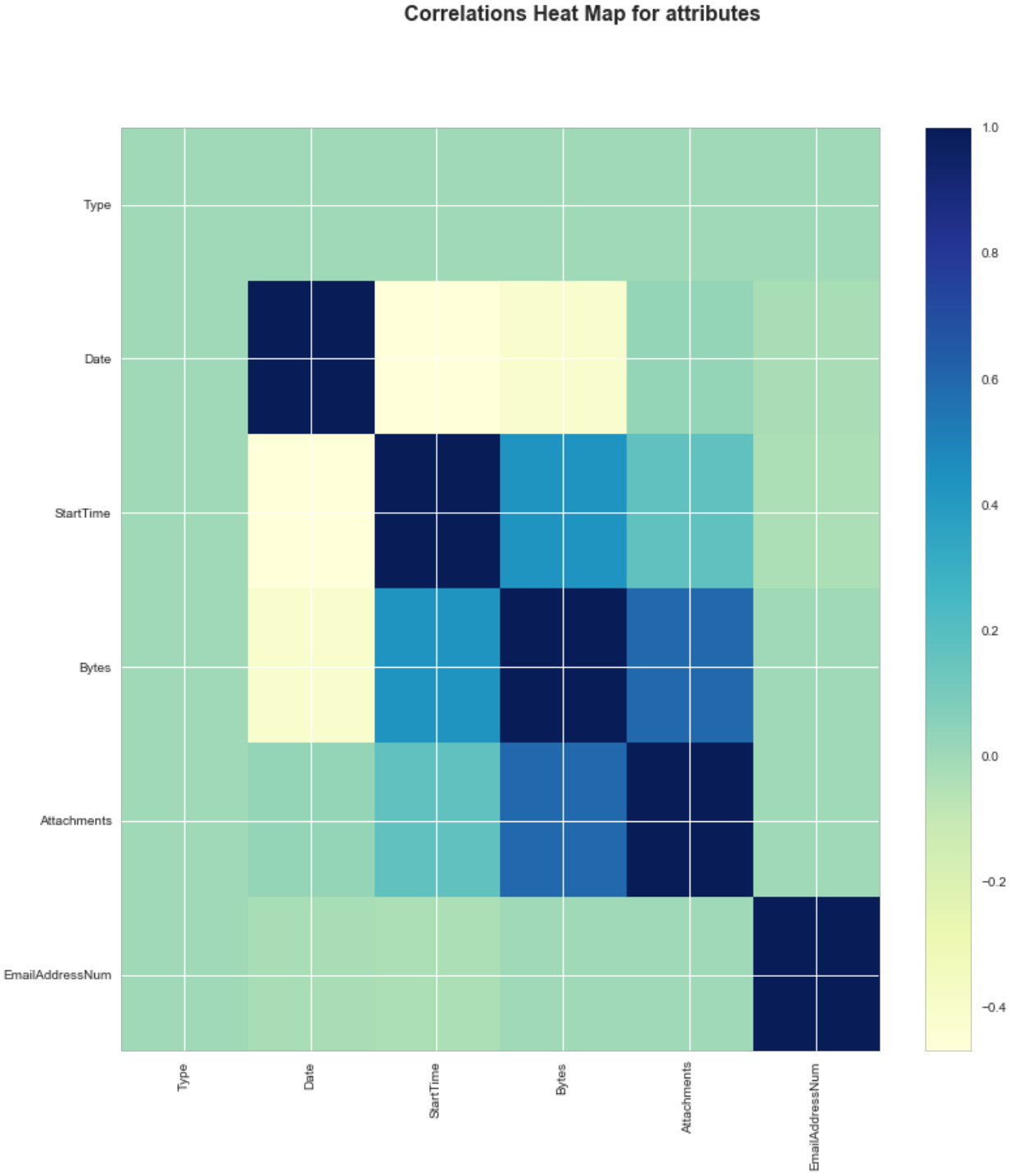




Here, we could see evenly distributed data among all email ids for 100 and 200 attatchments.

Statistics: Correlation:

In [97]:
```
corltn=type3.corr()
corltn=corltn.fillna(0)
plt.figure(figsize=(13, 13))
plt.imshow(corltn, cmap='YlGnBu', interpolation='none', aspect='auto')
plt.colorbar()
plt.xticks(range(len(corltn)), corltn.columns, rotation='vertical')
plt.yticks(range(len(corltn)), corltn.columns);
plt.suptitle(' Correlations Heat Map for attributes', fontsize=16, fontweight='bold')
```



Building the model: Linear Regression:

In [105]:
```
X_train, x_test, Y_train, y_test = train_test_split(
type3['UserNum'], type3['EmailAddressNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
X_train = X_train[:, np.newaxis]
Y_train = Y_train[:, np.newaxis]
x_test = x_test[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
plt.scatter(X_train, Y_train, color='navy', s=30, marker='o', label="training points")
plt.plot(x_test, y_predict, color='teal', linewidth=2)
plt.legend(loc='best')
plt.show()
```

Performance Evaluation:

In [106]:
```
r_data= type3[['UserNum','MachineNum','Date','Bytes','Attachments']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type3['EmailAddressNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

Out[106]: 1.5710523165197476

In [107]:
```
r_data= type3[['UserNum','MachineNum','Date','Bytes','Attachments','StartTime']]
X_train, x_test, Y_train, y_test = train_test_split(
r_data, type3['EmailAddressNum'], test_size=0.2, random_state=42)
regr = linear_model.LinearRegression()
Y_train = Y_train[:, np.newaxis]
regr.fit(X_train, Y_train)
y_predict = regr.predict(x_test)
mean_squared_error(y_test,y_predict)
```

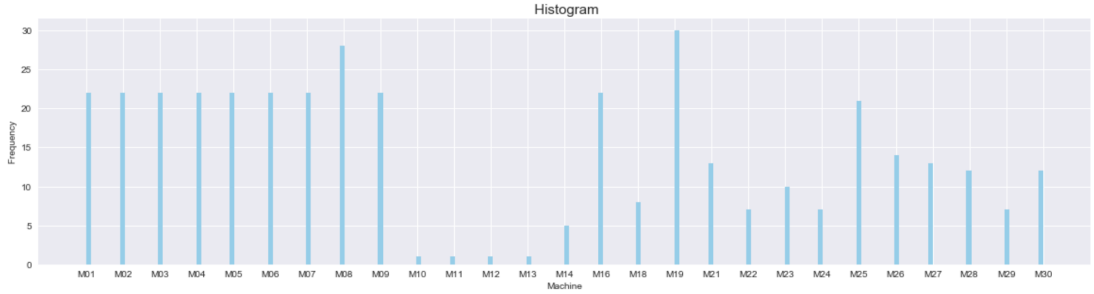
Out[107]: 1.6061310501077473

Here, we obtained good mean square error value and we could strongly recommend this model.

MACHINE USAGE PATTERN:

Machine: Histogram:

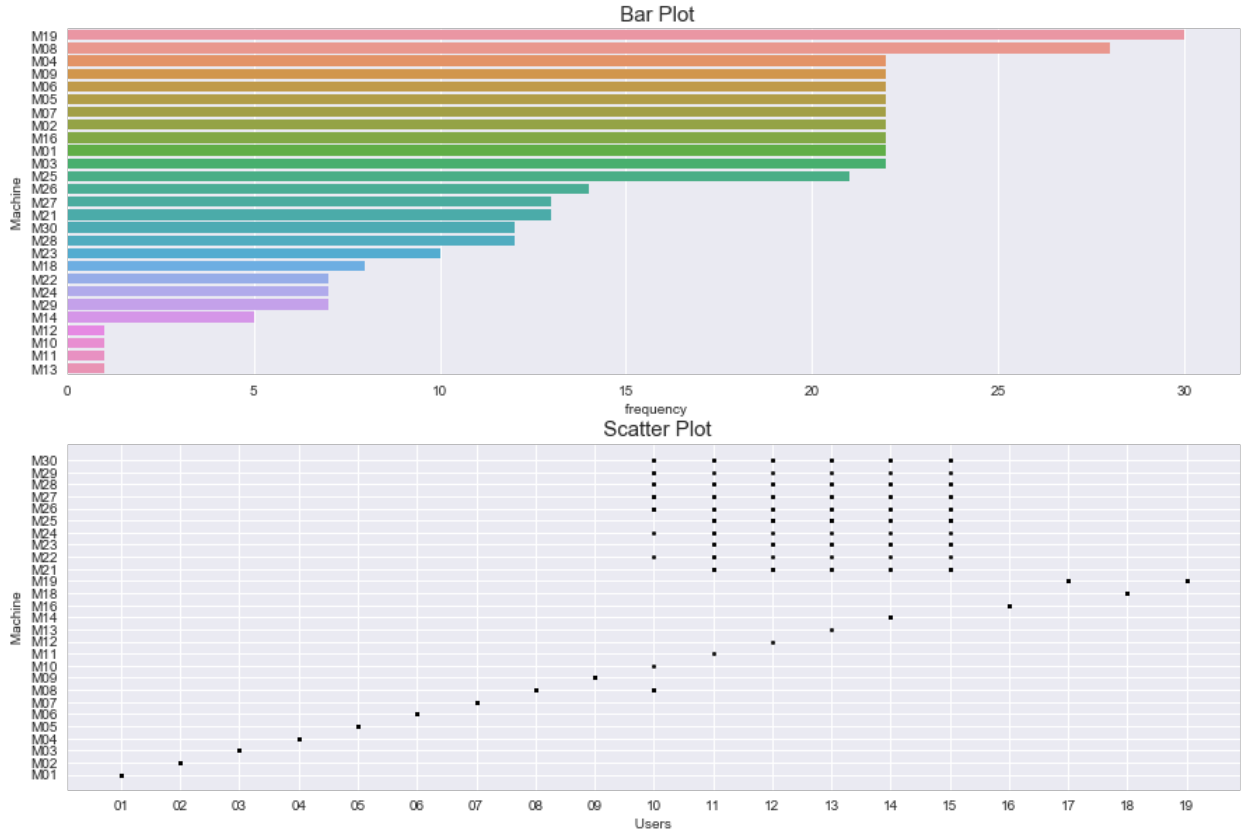
In [108]:
```
fig, ax = plt.subplots(1, 1, figsize = (20, 5))
ax.hist(type1.Machine, bins = 200, range = [min(type1.Machine), max(type1.Machine)], label = "price",color = "skyblue")
ax.set_title("\n \n  Histogram ", fontsize = 15)
ax.set_xlabel("Machine", fontsize = 10)
ax.set_ylabel(" Frequency ", fontsize = 10)
plt.show()
```



User:

```
In [109]: machine = type1["Machine"].value_counts()
          print("Unique Machine Names :", machine.size)
          fig, ax = plt.subplots(2, 1, figsize = (15, 10))
          sns.barplot(machine.values, machine.index,ax = ax[0])
          ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
          ax[0].set_xlabel(" Sale Price", fontsize = 10)
          f1 = type1['UserNum'].values
          f2 = type1['Machine'].values
          X = np.array(list(zip(f1, f2)))
          ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
          ax[1].scatter(f1, f2, c='black', s=7)
          ax[0].set_xlabel(" frequency", fontsize = 10)
          ax[0].set_ylabel(" Machine ", fontsize = 10)
          ax[1].set_xlabel(" Users ", fontsize = 10)
          ax[1].set_ylabel(" Machine ", fontsize = 10)
          plt.show()
```

```
Unique Machine Names : 27
```

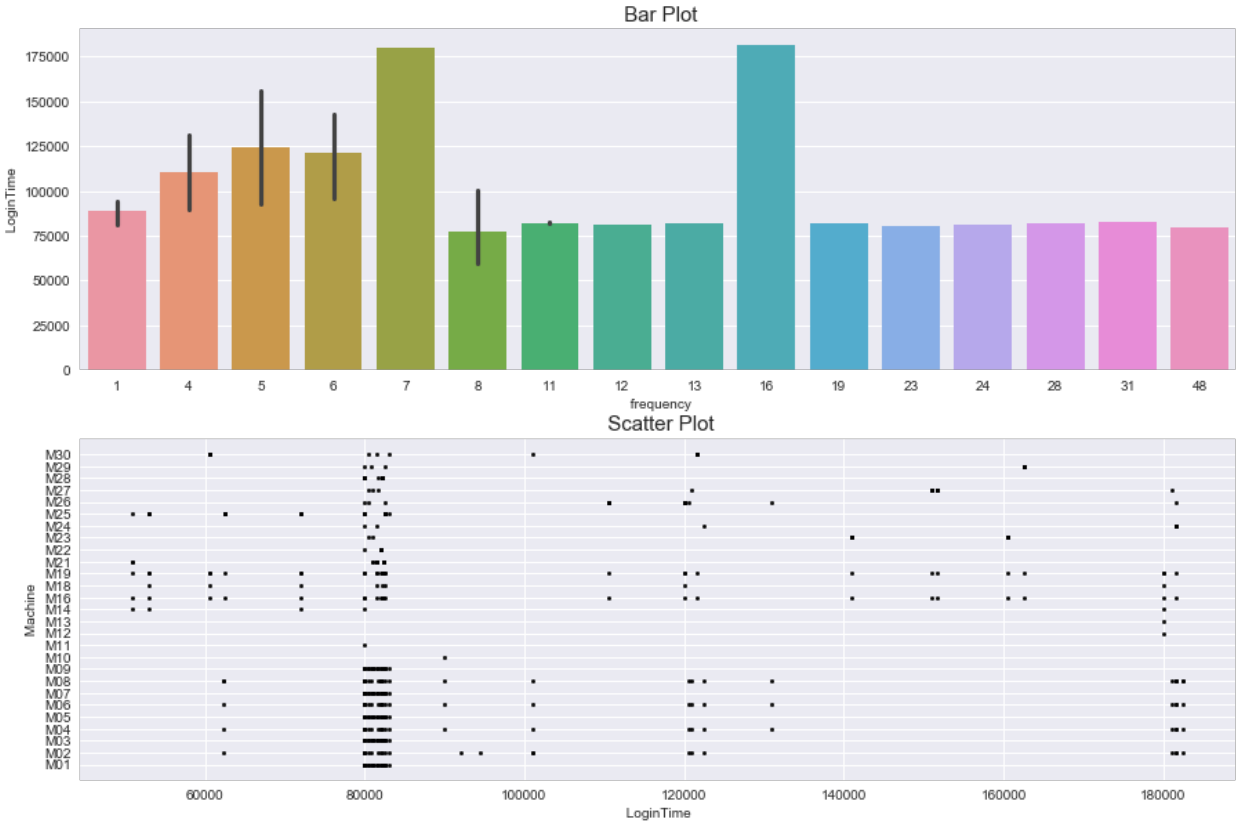


Here, we can understand that U10 to U15 are having similar pattern.

LoginTime: Bar plot and Scatter plot.

In [110]:
```
loginTime = type1["LoginTime"].value_counts()
print("Unique LoginTime :", loginTime.size)
fig, ax = plt.subplots(2, 1, figsize = (15, 10))
sns.barplot(loginTime.values, loginTime.index,ax = ax[0])
ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
ax[0].set_xlabel(" Sale Price", fontsize = 10)
f1 = type1['LoginTime'].values
f2 = type1['Machine'].values
X = np.array(list(zip(f1, f2)))
ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
ax[1].scatter(f1, f2, c='black', s=7)
ax[0].set_xlabel(" frequency", fontsize = 10)
ax[0].set_ylabel(" LoginTime ", fontsize = 10)
ax[1].set_xlabel(" LoginTime ", fontsize = 10)
ax[1].set_ylabel(" Machine ", fontsize = 10)
plt.show()
```

```
Unique LoginTime : 39
```

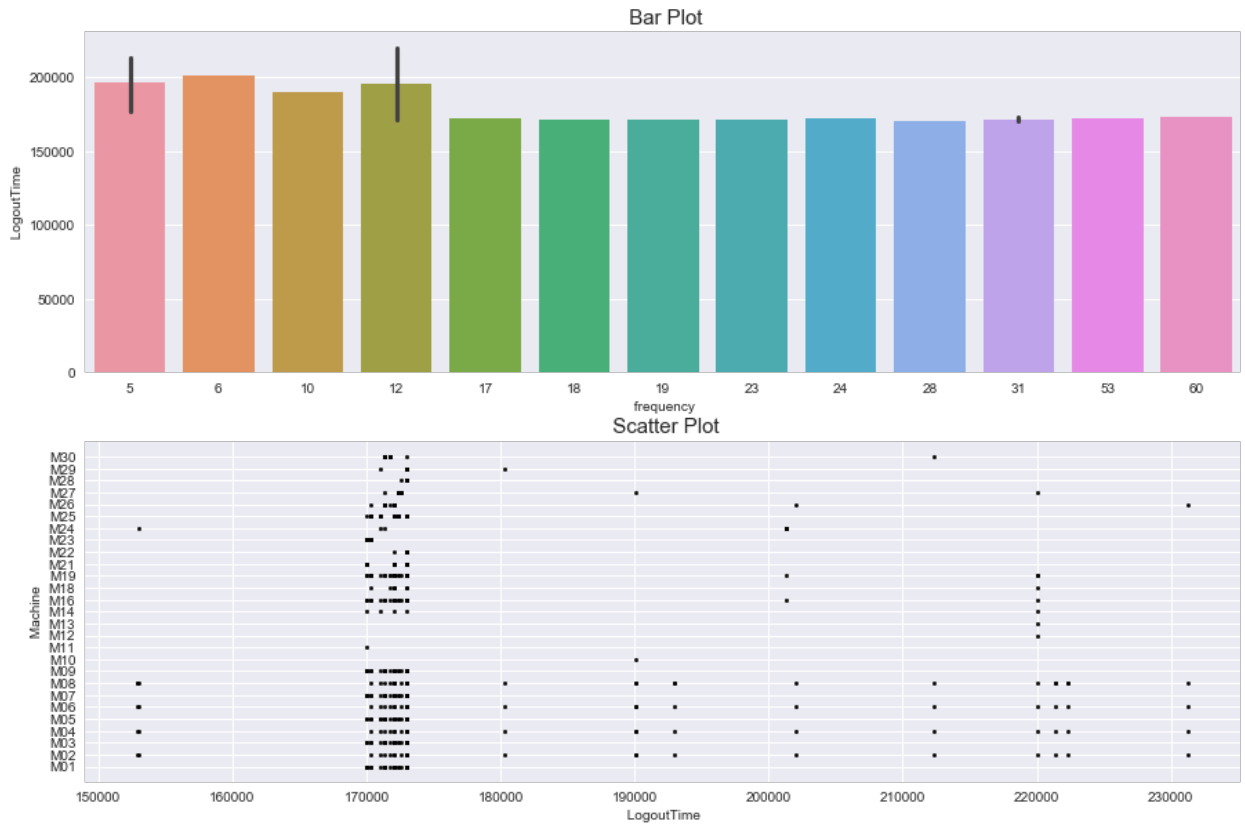


Here, we can infer, the login time is densily distributed at 8:00

LogoutTime: Bar plot and Scatter plot.

```
In [111]: logoutTime = type1["LogoutTime"].value_counts()
          print("Unique LogoutTime :", logoutTime.size)
          fig, ax = plt.subplots(2, 1, figsize = (15, 10))
          sns.barplot(logoutTime.values, logoutTime.index,ax = ax[0])
          ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
          ax[0].set_xlabel(" Sale Price", fontsize = 10)
          f1 = type1['LogoutTime'].values
          f2 = type1['Machine'].values
          X = np.array(list(zip(f1, f2)))
          ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
          ax[1].scatter(f1, f2, c='black', s=7)
          ax[0].set_xlabel(" frequency", fontsize = 10)
          ax[0].set_ylabel(" LogoutTime ", fontsize = 10)
          ax[1].set_xlabel(" LogoutTime ", fontsize = 10)
          ax[1].set_ylabel(" Machine ", fontsize = 10)
          plt.show()
```

```
Unique LogoutTime : 23
```

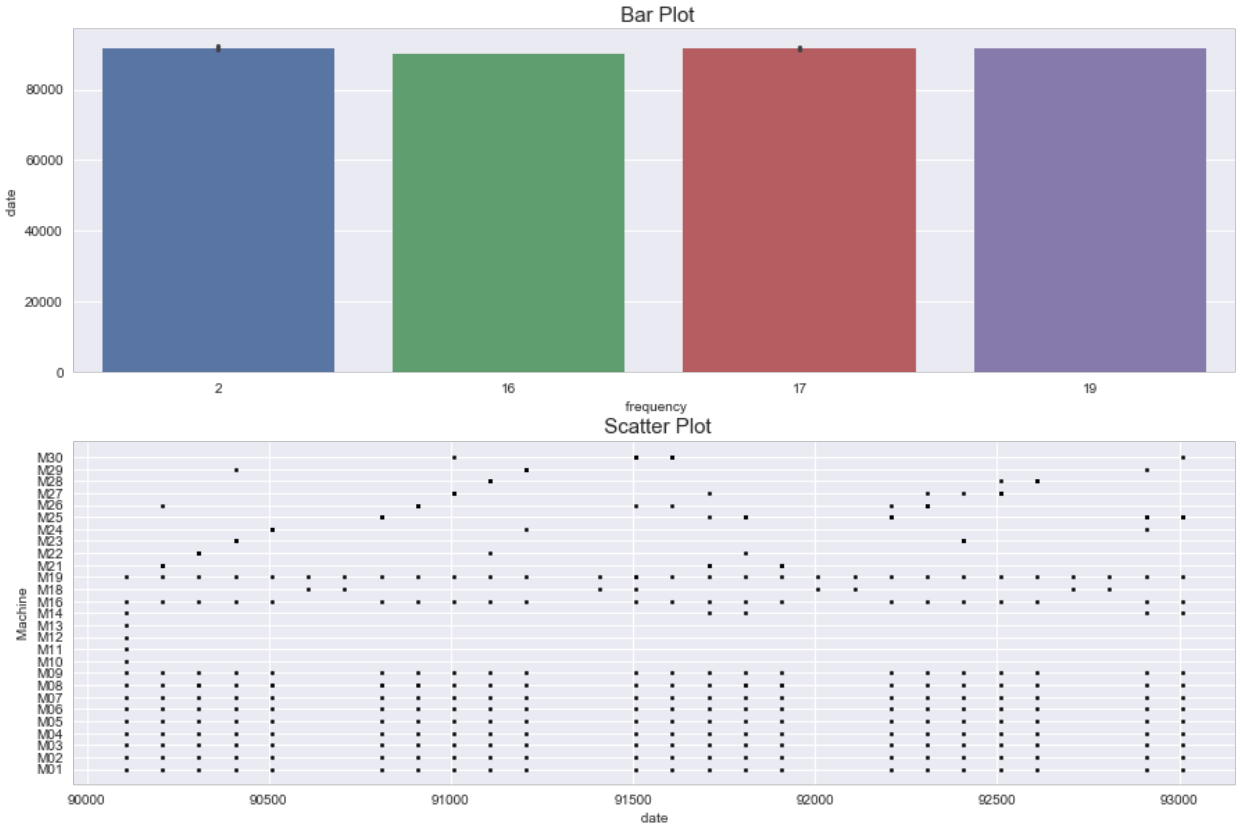


Here, we can infer the log out time is densely distributed at 17:00

Date: Bar plot and Scatter plot.

```
In [112]: date = type1["Date"].value_counts()
          print("Unique Date :", date.size)
          fig, ax = plt.subplots(2, 1, figsize = (15, 10))
          sns.barplot(date.values, date.index,ax = ax[0])
          ax[0].set_title("\n \n  Bar Plot ", fontsize = 15)
          ax[0].set_xlabel(" Sale Price", fontsize = 10)
          f1 = type1['Date'].values
          f2 = type1['Machine'].values
          X = np.array(list(zip(f1, f2)))
          ax[1].set_title("\n \n  Scatter Plot ", fontsize = 15)
          ax[1].scatter(f1, f2, c='black', s=7)
          ax[0].set_xlabel(" frequency", fontsize = 10)
          ax[0].set_ylabel(" date ", fontsize = 10)
          ax[1].set_xlabel(" date ", fontsize = 10)
          ax[1].set_ylabel(" Machine ", fontsize = 10)
          plt.show()
```

```
Unique Date : 29
```



Here, we can infer that from M01 to M1o there exists similar behaviour on all days.

Building the model:Linear Regression:

```
In [114]: X_train, x_test, Y_train, y_test = train_test_split(
          type1['UserNum'], type1['MachineNum'], test_size=0.2, random_state=42)
          regr = linear_model.LinearRegression()
          X_train = X_train[:, np.newaxis]
          Y_train = Y_train[:, np.newaxis]
          x_test = x_test[:, np.newaxis]
          regr.fit(X_train, Y_train)
          y_predict = regr.predict(x_test)
          plt.scatter(X_train, Y_train, color='navy', s=30, marker='o', label="training points")
          plt.plot(x_test, y_predict, color='teal', linewidth=2)
          plt.legend(loc='best')
          plt.show()
```

Performance evaluation:

```
In [115]: r_data= type1[['UserNum','Date','LoginTime','LogoutTime','AvgUserProcess','MaxUserProcess']]
          X_train, x_test, Y_train, y_test = train_test_split(
          r_data, type1['MachineNum'], test_size=0.2, random_state=42)
          regr = linear_model.LinearRegression()
          Y_train = Y_train[:, np.newaxis]
          regr.fit(X_train, Y_train)
          y_predict = regr.predict(x_test)
          mean_squared_error(y_test,y_predict)

Out[115]: 26.19339116475911
```

Here, it results in a decent mean square error value and we could consider this model for finding patterns.

USER PROFILING:

LOGIN ACCESS PATTERN:

Association rules:

1) If Login time=08:00 , Logout time= 17:00 then No.of login days=22
   coverage=6, support=6, Accuracy=100%
2) If Times at which the user logged out before/after working hours= 20:10,22:00
Times at which the user logged in before/after working hours= 5:00, 6:00, 7:00, 18:00
Then, Login time= More than once
   coverage=5, support=5, Accuracy=100%
3) If Times at which the user logged out before/after working hours= 20:10,22:00
Times at which the user logged in before/after working hours= 5:00, 6:00, 7:00, 18:00
Then, Logout Time= More than once
   coverage=5, support=5, Accuracy=100%

User	Machine	No of Login days	Login Time	Logout Time	# days Logged in during Weekdays	# days logged in during weekends	Times at which user logged in before/after working hours	Times at which user logged out before/after working hours
U01	M01	22	8:00	17:00	22	0	0	0
U02	M02	22	More than once	More than once	22	0	06:00, 18:00	18:00,19:00,19:15,20:15,21:15,22:15,23:15
U03	M03	22	8:00	17:00	22	0	0	0
U04	M04	22	More than once	More than once	22	0	06:00, 18:00	18:00,19:00,19:15,20:15,21:15,22:15,23:15
U05	M05	22	8:00	17:00	22	0	0	0
U06	M06	22	More than once	More than once	22	0	06:00, 18:00	18:00,19:00,19:15,20:15,21:15,22:15,23:15
U07	M07	22	8:00	17:00	22	0	0	0
U08	M08	22	More than once	More than once	22	0	06:00, 18:00	18:00,19:00,19:15,20:15,21:15,22:15,23:15
U09	M09	22	8:00	17:00	22	0	0	0
U10	M08,M10,M22,24,M26,M27,M28,M29,M30	22	More than once	More than once	22	0	06:00, 18:00	18:00,19:00,19:15,20:15,21:15,22:15,23:15
U11	M11,M21,M22,24,M26,M27,M28,M29,M30	22	8:00	17:00	22	0	0	0
U12	M12,M21,M22,24,M26,M27,M28,M29,M30	22	More than once	More than once	22	0	5:00,6:00,7:00,18:00	20:10,22:00
U13	M13,M21,M22,24,M26,M27,M28,M29,M30	22	More than once	More than once	22	0	5:00,6:00,7:00,18:00	20:10,22:00
U14	M14,M21,M22,24,M26,M27,M28,M29,M30	22	More than once	More than once	22	0	5:00,6:00,7:00,18:00	20:10,22:00
U15	M21,M22,24,M26,M27,M28,M29,M30	21	More than once	More than once	21	0	5:00,6:00,7:00,18:00	20:10
U16	M16	22	More than once	More than once	22	0	5:00,6:00,7:00,18:00	20:10,22:00
U17	M19	22	More than once	More than once	22	0	5:00,6:00,7:00,18:00	20:10,22:00
U18	M18	8	More than once	More than once	1	7	5:00,6:00,7:00,18:00	22:00
U19	M19	8	More than once	More than once	1	7	5:00,6:00,7:00,18:00	22:00

PROGRAM ACCESS PATTERN:

Program #	User	Machine	Execution Time	Start Time	Frequency
LP010	U01,U07,U13,U16,U17,U19	M01,M07,M16,M19,M21	250	Multiple values	18
LP020	U03,U10,U17,U19	M03,M08,M19,M26	Multiple values	Multiple values	15
LP050	U01,U07,U13,U16,U17,U19	M01,M07,M16,M19,M21	Multiple values	Multiple values	20
LP060	U03,U10,U17,U19	M03,M08,M19,M26	Multiple values	Multiple values	15
LP075	U09	M09	1100	Multiple values	7
LP080	U01,U07	M01,M07	400	Multiple values	10
LP085	U09	M09	400	Multiple values	5
LP090	U03,U10,U17,U19	M03,M08,M19,M28		Multiple values	10
LP095	U09	M09	250	Multiple values	6
UP010	U02,U04,U06,U08,U10	M02,M04,M06,M08,M10,M23,M26,M29	250	Multiple values	29
UP029	U18	M18	1100	16:30	1
UP082	U19	M18	250	16:00	1
UP111	U20	M18	250	9:00	1
UP134	U21	M18	1100	11:30	1
UP150	U02,U04,U06,U08,U10	M02,M04,M06,M08,M10,M28	1100	Multiple values	34
UP170	U02,U04,U06,U08,U10	M02,M04,M06,M08,M10,M30	400	Multiple values	23
UP290	U18	M18	400	Multiple values	1
UP300	U02,U04,U06,U08,U10	M02,M04,M06,M08,M23	Multiple values	Multiple values	18
UP310	U05,U11,U12,U13,U14,U15,U16	M05,M11,M12,M13,M14,M15,M16,M22, M24,M25,M28	250	Multiple values	39
UP350	U02,U04,U06,U08,U10,U11,U12 ,U13,U14,U15,U16	M02,M04,M06,M08,M11,M12,M13,M14, M15,M16,M22,M24,M25,M28	650	Multiple values	70
UP380	U05,U11,U12,U13,U14,U15,U16	M05,M11,M12,M13,M14,M15,M16,M22, M24,M25	400	Multiple values	32
UP400	U18	M18	1100	12:30	1
UP420	U18	M18	250	16:00	1
UP463	U18	M18	850	16:15	1

Association rule:

1) If User= U18 and Machine =M18
   Then, Frequency=1
   coverage=4, support=4, Accuracy=100%

FILE ACCESS PATTERN:

File #	User	Machine	Status
F0010	U05,U06,U07,U08,U09, U10	M05,M06,M07,M08,M09,M10,M23,M26,M27 ,M28,M29	R,RW
F0019	U16,U17,U18,U19	M16,M18,M19	RW
F0020	U05,U06,U07,U08,U09, U10	M05,M06,M07,M08,M09,M10,M23,M26,M28 ,M30	R,RW
F0025	U05,U06,U07,U08,U09, U10	M05,M06,M07,M08,M09,M10,M21,M26	R,RW
F0059	U01,U02,U03,U04,U13	M01,M02,M03,M04,M21	R
F0070	U01,U02,U03,U04	M01,M02,M03,M04	
F0079	U01,U02,U03,U04,U13	M01,M02,M03,M04,M21	R
F0085	U01,U02,U03,U04	M01,M02,M03,M04	R
F0099	U16,U17,U18,U19	M16,M18,M19	R
F0100	U11,U12,13,14,U15	M11,M12,M13,M14,M15,M22,M24,M25,M28	R
F0111	U16,U17,U18,U19	M16,M18,M19	R
F0159	U01,U02,U03,U04	M01,M02,M03,M04	R
F0170	U01,U02,U03,U04	M01,M02,M03,M04	RW
F0185	U01,U02,U03,U04	M01,M02,M03,M04	R,RW
F0200	U11,U12,13,14,U15,U16 ,U17,U18,U19	M11,M12,M13,M14,M15,M16,M18,M19,M22 ,M24,M25,M28	R
F0222	U16,U17,U18,U19	M16,M18,M19	R
F0270	U01,U02,U03,U04	M01,M02,M03,M04	RW
F0277	U16,U17	M16,M19	R
F0285	U02,U04	M02,M04	R
F0300	U11,U12,13,14,U15	M11,M12,M13,M14,M15,M22,M24,M25	R
F0333	U16,U17,U18,U19	M16,M18,M19	R
F0337	U16,U17	M16,M19	R
F0385	U01,U02,U03,U04	M01,M02,M03,M04	R
F0389	U01,U02,U03,U04	M01,M02,M03,M04	R
F0444	U16,U17,U18,U19	M16,M18,M19	R
F0447	U16,U17	M16,M19	R
F0471	U01,U02,U03,U04	M01,M02,M03,M04	R
F0475	U01,U02,U03,U04	M01,M02,M03,M04	RW
F0555	U16,U17,U18,U19	M16,M18,M19	R

Association rule:

1) If User= U01,U02,U03,U04 and Machine =M01,M02,M03,M04
Then, Status=RW, coverage=10, support=5, Accuracy=50%

As the accuracy is falling under 75%, this cannot be made as a Rule.

```
In [240]: def divide_cats(data):
              if( data == "U01"or data=="U02"or data == "U03" or data== "U04"):
                  return "File 59-File 2000"
              if( data == "U05"or data=="U06"or data == "U07" or data== "U08"or data == "U09" or data== "U10"):
                  return "File 0-File 25"
              if( data == "U11"or data=="U12"or data == "U13" or data== "U14"or data == "U15"):
                  return "File 50-File 300"
              if( data == "U16"or data=="U17"or data == "U18" or data== "U19"):
                  return "File 10-File 555"
              return "CATOTHER"

In [241]: mercariframe['file_cats'] =mercariframe.User.map(lambda x : divide_cats(x))

In [360]: mercariframe.head()
```

Out[360]:

te	StartTime	Program	ExecutionTime	File	Status	Printer	PagesPrinted	UserNum	MachineNum	ProgramNum	FileNum	StatusNum	PrinterNum	file_cats
)8	90201	LP010	340	F0059	R	PR1	10.0	01	01	0	4	0	0	File 59-File 2000
)8	90201	LP020	340	F0059	R	PR1	10.0	03	03	1	4	0	0	File 59-File 2000
)8	90201	UP310	340	F0010	RW	PR2	10.0	05	05	18	0	1	1	File 0-File 25
)8	90201	LP010	340	F0010	RW	PR2	10.0	07	07	0	0	1	1	File 0-File 25
)8	90201	LP095	340	F0010	RW	PR2	10.0	09	09	8	0	1	1	File 0-File 25

```
In [252]: type1.head()
```

Added a new column file_cats and divided the users to different categories according to file usage.

PRINTER USAGE PATTERN:

Printer	User	Machine	File	Pages printed	Frequency
PR1	U01,U02,U03,U04	M01,M02,M03,M04	F0059,F0070,F0079,F0085,F0159,F0170,F0185,F0270,F0285,F0385,F0389,F0471,F0475	1,2,3,4,5,7,10,12,15,20,21	90
PR2	U05,U06,U07,U08,U09,U10	M05,M06,M07,M08,M09,M10,M21,M23,M26,M27,M28,M29,M30	F0010,F0020,F0025	1,2,3,4,5,7,10,12,15,20,21	135
PR3	U11,U12	M11,M12,M22,M24,M25,M28	F0100,F0200,F0300	1,2,3,4,5,7,10,12,15,20,21	35
PR4	U13,U14,U15,U16	M13,M14,M15,M16,M22,M24,M25,M28	F0019,F0059,F0079,F0099,F0100,F0111,F0200,F0222,FO277,F0300,F0333,F0337,F0444,F0447,F0555	1,2,3,4,5,7,10,12,15,20,21	70
PR5	U18	M18	F0019,F0099,F0100,F0111,F0200,F0222,F0333,F0444,F0555	3,4,5,7,10,12	5
PR6	U17,U19	M19	F0019,F0099,F0100,F0111,F0200,F0222,F0277,F0333,F0337,F0444,F0447,F0555	3,4,5,7,10,12,21	20

EMAIL PATTERN:

Email	User	Machine	Email program	Status	Attachments	Bytes
bob@xyz.com	U15,U16,U17,U18,U19	M15,M16,M18,M19,M22,M25,M28	E1,E4,E5	RECEIVE	0,1,2	0-3203045
jones@pqr.com	U01,U02,U03,U04,U05	M01,M02,M03,M04,M05	E1	SEND	0,1,2	0-3203045
mom@icare.com	U02,U04,U06,U08,U10	M02,M04,M06,M08	E1,E3	RECEIVE, SEND	0	0
smith@abc.org	U06,U07,U08,U09,U10	M06,M07,M08,M09,M24,M25,M30	E1,E3	RECEIVE, SEND	0,1,2	0-3203045
xyz@sai.org	U11,U12,U13,U14	M11,M12,M13,M14,M22,M25,M28	E1	SEND	0,1,2	0-3203045

Function that combines user on the condition they send/receive an email from specific person

In [334]:
```
def email_category(data):
    return type3.loc[type3['EmailAddress'] == data]
```

In [132]:
```
def email_status(data):
    return gtrain_cats.loc[gtrain_cats['Status'] == data]
```

In [335]:
```
gtrain_cats= email_category('smith@abc.org')
# print(gtrain_cats.shape)
gtrain_cats.head()
```

Out[335]:

	Type	User	Machine	Date	StartTime	EmailProgram	EmailAddress	Status	Bytes	Attachments	EmailAddressNum	UserNum	MachineNum
4	3	U07	M07	90108	120656	E1	smith@abc.org	S	209003	1	3	07	07
5	3	U09	M09	90108	120656	E3	smith@abc.org	S	209003	1	3	09	09
8	3	U06	M06	90108	140656	E1	smith@abc.org	R	209003	1	3	06	06
9	3	U08	M08	90108	140656	E1	smith@abc.org	R	209003	1	3	08	08
10	3	U10	M24	90108	140656	E3	smith@abc.org	R	209003	1	3	10	24

In [336]:
```
gtrain_status= email_status('S')
# print(gtrain_cats.shape)
gtrain_status
```

Out[336]:

	Type	User	Machine	Date	StartTime	EmailProgram	EmailAddress	Status	Bytes	Attachments	EmailAddressNum	UserNum	MachineNum
4	3	U07	M07	90108	120656	E1	smith@abc.org	S	209003	1	3	07	07
5	3	U09	M09	90108	120656	E3	smith@abc.org	S	209003	1	3	09	09
15	3	U07	M07	90108	150302	E1	smith@abc.org	S	3203045	2	3	07	07
16	3	U09	M09	90108	150302	E3	smith@abc.org	S	3203045	2	3	09	09
49	3	U07	M07	90808	100656	E1	smith@abc.org	S	2090	0	3	07	07
51	3	U09	M09	90808	100656	E3	smith@abc.org	S	2090	0	3	09	09
66	3	U07	M07	90808	160302	E1	smith@abc.org	S	3203	0	3	07	07
68	3	U09	M09	90808	160302	E3	smith@abc.org	S	3203	0	3	09	09
87	3	U07	M07	91808	100656	E1	smith@abc.org	S	132090	1	3	07	07
89	3	U09	M09	91808	100656	E3	smith@abc.org	S	132090	1	3	09	09
104	3	U07	M07	91808	160302	E1	smith@abc.org	S	456780	1	3	07	07
106	3	U09	M09	91808	160302	E3	smith@abc.org	S	456780	1	3	09	09
125	3	U07	M07	92608	100656	E1	smith@abc.org	S	1405	0	3	07	07

In [337]:
```
gtrain_statusreceived= email_status('R')
# print(gtrain_cats.shape)
gtrain_statusreceived
```

Out[337]:

	Type	User	Machine	Date	StartTime	EmailProgram	EmailAddress	Status	Bytes	Attachments	EmailAddressNum	UserNum	MachineNum
8	3	U06	M06	90108	140656	E1	smith@abc.org	R	209003	1	3	06	06
9	3	U08	M08	90108	140656	E1	smith@abc.org	R	209003	1	3	08	08
10	3	U10	M24	90108	140656	E3	smith@abc.org	R	209003	1	3	10	24
19	3	U06	M06	90108	153302	E1	smith@abc.org	R	3203045	2	3	06	06
20	3	U08	M08	90108	153302	E1	smith@abc.org	R	3203045	2	3	08	08
21	3	U10	M25	90108	153302	E3	smith@abc.org	R	3203045	2	3	10	25

In [338]:
```
gtrain_cats= email_category('jones@pqr.com')
# print(gtrain_cats.shape)
gtrain_cats.head()
```

Out[338]:

	Type	User	Machine	Date	StartTime	EmailProgram	EmailAddress	Status	Bytes	Attachments	EmailAddressNum	UserNum	MachineNum
0	3	U01	M01	90108	120656	E1	jones@pqr.com	S	209003	1	1	01	01
1	3	U02	M02	90108	120656	E1	jones@pqr.com	S	209003	1	1	02	02
2	3	U03	M03	90108	120656	E1	jones@pqr.com	S	209003	1	1	03	03
3	3	U05	M05	90108	120656	E1	jones@pqr.com	S	209003	1	1	05	05
7	3	U04	M04	90108	140656	E1	jones@pqr.com	S	209003	1	1	04	04

In [339]:
```
gtrain_statusreceived= email_status('R')
# print(gtrain_cats.shape)
gtrain_statusreceived
```

Out[339]:

Type	User	Machine	Date	StartTime	EmailProgram	EmailAddress	Status	Bytes	Attachments	EmailAddressNum	UserNum	MachineNum

In [340]:
```
gtrain_statusreceived= email_status('S')
# print(gtrain_cats.shape)
gtrain_statusreceived
```

Out[340]:

	Type	User	Machine	Date	StartTime	EmailProgram	EmailAddress	Status	Bytes	Attachments	EmailAddressNum	UserNum	MachineNum
0	3	U01	M01	90108	120656	E1	jones@pqr.com	S	209003	1	1	01	01
1	3	U02	M02	90108	120656	E1	jones@pqr.com	S	209003	1	1	02	02
2	3	U03	M03	90108	120656	E1	jones@pqr.com	S	209003	1	1	03	03
3	3	U05	M05	90108	120656	E1	jones@pqr.com	S	209003	1	1	05	05
7	3	U04	M04	90108	140656	E1	jones@pqr.com	S	209003	1	1	04	04
11	3	U01	M01	90108	150302	E1	jones@pqr.com	S	3203045	2	1	01	01
12	3	U02	M02	90108	150302	E1	jones@pqr.com	S	3203045	2	1	02	02
13	3	U03	M03	90108	150302	E1	jones@pqr.com	S	3203045	2	1	03	03
14	3	U05	M05	90108	150302	E1	jones@pqr.com	S	3203045	2	1	05	05
18	3	U04	M04	90108	153302	E1	jones@pqr.com	S	3203045	2	1	04	04
43	3	U01	M01	90808	100656	E1	jones@pqr.com	S	2090	0	1	01	01
44	3	U02	M02	90808	100656	E1	jones@pqr.com	S	2090	0	1	02	02
45	3	U03	M03	90808	100656	E1	jones@pqr.com	S	2090	0	1	03	03
46	3	U04	M04	90808	100656	E1	jones@pqr.com	S	2090	0	1	04	04
47	3	U05	M05	90808	100656	E1	jones@pqr.com	S	2090	0	1	05	05
60	3	U01	M01	90808	160302	E1	jones@pqr.com	S	3203	0	1	01	01
61	3	U02	M02	90808	160302	E1	jones@pqr.com	S	3203	0	1	02	02
62	3	U03	M03	90808	160302	E1	jones@pqr.com	S	3203	0	1	03	03
63	3	U04	M04	90808	160302	E1	jones@pqr.com	S	3203	0	1	04	04
64	3	U05	M05	90808	160302	E1	jones@pqr.com	S	3203	0	1	05	05
81	3	U01	M01	91808	100656	E1	jones@pqr.com	S	132090	1	1	01	01
82	3	U02	M02	91808	100656	E1	jones@pqr.com	S	132090	1	1	02	02
83	3	U03	M03	91808	100656	E1	jones@pqr.com	S	132090	1	1	03	03
84	3	U04	M04	91808	100656	E1	jones@pqr.com	S	132090	1	1	04	04
85	3	U05	M05	91808	100656	E1	jones@pqr.com	S	132090	1	1	05	05
98	3	U01	M01	91808	160302	E1	jones@pqr.com	S	456780	1	1	01	01
99	3	U02	M02	91808	160302	E1	jones@pqr.com	S	456780	1	1	02	02

Here there is no data displayed for jones@pqr.com since users don't receive any email from jones.

MACHINE USAGE PATTERN:

Machine	User	Machine used during weekdays	Machine used during weekends	Frequent login time	Frequent logout time	Total CPU usage
M01	U01	Yes	No	8:00	17:00	241844
M02	U02	Yes	No	8:00	17:00	241844
M03	U03	Yes	No	8:00	17:00	241844
M04	U04	Yes	No	8:00	17:00	241844
M05	U05	Yes	No	8:00	17:00	241844
M06	U06	Yes	No	8:00	17:00	241844
M07	U07	Yes	No	8:00	17:00	241844
M08	U08,U10	Yes	No	8:00	17:00	309310
M09	U09	Yes	No	8:00	17:00	241844
M10	U10	Yes	No	9:00	19:00	12098
M11	U11	Yes	No	8:00	17:00	12098
M12	U12	Yes	No	18:00	22:00	12098
M13	U13	Yes	No	18:00	22:00	12098
M14	U14	Yes	No	Before 8:00	17:00	241844
M16	U16	Yes	No	multiple values	17:00	241844
M18	U18	Yes	Yes	multiple values	17:00	84650
M19	U17	Yes	Yes	multiple values	17:00	326494
M21	U11,U12,U13,U14,U15	Yes	No	8:00	17:15	110844
M22	U10,U11,U12,U13,U14,U15	Yes	No	8:00	17:15	82700
M23	U11,U12,U13,U14,U15	Yes	No	8:00	17:15	146660
M24	U10,U11,U12,U13,U14,U15	Yes	No	8:00	17:15	85586
M25	U11,U12,U13,U14,U15	Yes	No	multiple values	17:15	228112
M26	U10,U11,U12,U13,U14,U15	Yes	No	multiple values	17:15	172976
M27	U10,U11,U12,U13,U14,U15	Yes	No	8:00	17:15	51116
M28	U10,U11,U12,U13,U14,U15	Yes	No	8:00	17:15	150490
M29	U10,U11,U12,U13,U14,U15	Yes	No	8:00	17:15	104756
M30	U10,U11,U12,U13,U14,U15	Yes	No	8:00	17:15	149410

CLUSTERING: K-Means: it is an unsupervised algorithm to find patterns or grouping in un-labeled data.

Login access data:

```
=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 2
Within cluster sum of squared errors: 38.04552590266875

Initial starting points (random):

Cluster 0: U10,Multiple,22,Multiple,Multiple,22,0,2,7
Cluster 1: U07,M07,22,8:00,17:00,22,0,0,0

Missing values globally replaced with mean/mode

Final cluster centroids:
                                                          Cluster#
Attribute                                      Full Data          0          1
                                                  (19.0)     (13.0)      (6.0)
=============================================================================================
User                                                 U01        U02        U01
Machine                                          Multiple   Multiple        M01
No of Login days                                  20.4737    19.7692         22
Login Time                                       Multiple   Multiple       8:00
Logout Time                                      Multiple   Multiple      17:00
# days Logged in during Weekdays                 19.7368    18.6923         22
# days logged in during weekends                  0.7368     1.0769          0
Times at which user logged in before/after working hours  2.2105     3.2308          0
Times at which user logged out before/after working hours  2.3684     3.4615          0



Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      13 ( 68%)
1       6 ( 32%)
```

Program pattern:

```
=== Run information ===

Scheme:       weka.clusterers.HierarchicalClusterer -N 2 -L SINGLE -P -A "weka.core.EuclideanDistance -R first-last"
Relation:     dm
Instances:    24
Attributes:   6
              Program #
              User
              Machine
              Execution Time
              Start Time
              Frequency
Test mode:    evaluate on training data


=== Clustering model (full training set) ===

Cluster 0
((((((250:1.00042,Multiple values:1.00042):0.73163,((250:1.41607,1100:1.41607):0.00082,(400:1.41607,Multiple values:1.41607



Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      23 ( 96%)
1       1 (  4%)
```

File access pattern:

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10
Relation:     dm
Instances:    29
Attributes:   4
              File #
              User
              Machine
              Status
Test mode:    evaluate on training data


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 2
Within cluster sum of squared errors: 59.0

Initial starting points (random):

Cluster 0: F0389,'U01,U02,U03,U04','M01,M02,M03,M04',R
Cluster 1: F0099,'U16,U17,U18,U19','M16,M18,M19',R

Missing values globally replaced with mean/mode

Final cluster centroids:
                                                        Cluster#
Attribute                                  Full Data            0            1
                                            (29.0)       (22.0)       (7.0)
=============================================================================
File #                                        F0010        F0010        F0019
User                                  U01,U02,U03,U04 U01,U02,U03,U04 U16,U17,U18,U19
Machine                               M01,M02,M03,M04 M01,M02,M03,M04     M16,M18,M19
Status                                             R            R            R
```

Printer Usage pattern:

```
kMeans
======

Number of iterations: 2
Within cluster sum of squared errors: 18.19748520710059

Initial starting points (random):

Cluster 0: PR4,'U13,U14,U15,U16','M13,M14,M15,M16,M22,M24,M25,M28','F0019,F0059,F0079,F0099,F0100,F0111,F0200,F0222,F0277,F0300,F0333,F0337,F0444,F0447,F0555','1,2,3,4,5,7,10,12,15,20,21',70
Cluster 1: PR1,'U01,U02,U03,U04','M01,M02,M03,M04','F0059,F0070,F0079,F0085,F0159,F0170,F0185,F0270,F0285,F0385,F0389,F0471,F0475','1,2,3,4,5,7,10,12,15,20,21',90

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute                                                                                          Full Data
                                                                                                       (6.0)
=======================================================================================================================
Printer                                                                                                  PR1
User                                                                                         U01,U02,U03,U04
Machine                                                                                      M01,M02,M03,M04
File                                   F0059,F0070,F0079,F0085,F0159,F0170,F0185,F0270,F0285,F0385,F0389,F0471,F0475
Pages printed                                                                    1,2,3,4,5,7,10,12,15,20,21
Frequency                                                                                            59.1667



Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      4 ( 67%)
1      2 ( 33%)
```

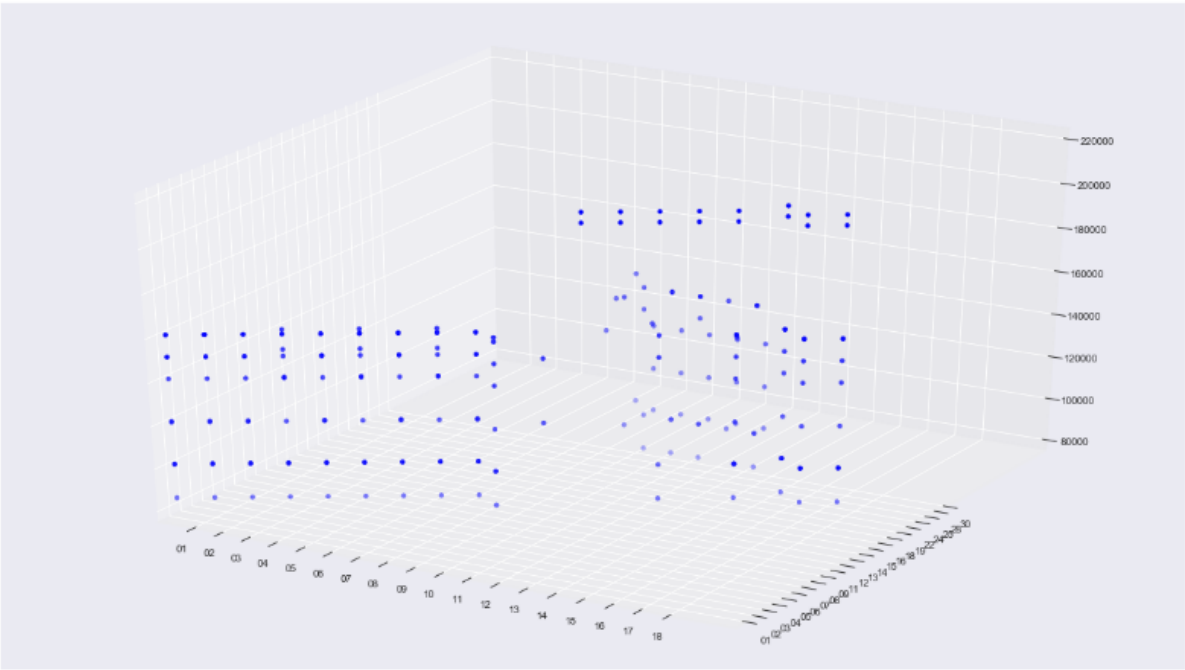
Email pattern:

```
kMeans
======

Number of iterations: 2
Within cluster sum of squared errors: 13.000000000000002

Initial starting points (random):

Cluster 0: smith@abc.org,'U06,U07,U08,U09,U10','M06,M07,M08,M09,M24,M25,M30','E1,E3','RECEIVE, SEND','0,1,2',0-3203045
Cluster 1: jones@pqr.com,'U01,U02,U03,U04,U05','M01,M02,M03,M04,M05',E1,SEND,'0,1,2',0-3203045

Missing values globally replaced with mean/mode

Final cluster centroids:
                                                                   Cluster#
Attribute                                   Full Data                     0                     1
                                                (5.0)                 (3.0)                 (2.0)
=================================================================================================
Email                                     bob@xyz.com           bob@xyz.com          jones@pqr.com
User                                  U15,U16,U17,U18,U19   U15,U16,U17,U18,U19   U01,U02,U03,U04,U05
Machine                       M15,M16,M18,M19,M22,M25,M28 M15,M16,M18,M19,M22,M25,M28   M01,M02,M03,M04,M05
Email program                                      E1                 E1,E3                    E1
Status                                           SEND         RECEIVE, SEND                  SEND
Attachments                                     0,1,2                 0,1,2                 0,1,2
Bytes                                       0-3203045             0-3203045             0-3203045



Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      3 ( 60%)
1      2 ( 40%)
```

K-means Clustering using Python:

Here we plotted user, machine and login time in a 3-D plot.

```
: fig = plt.figure()
  ax = Axes3D(fig)
  ax.scatter(demo1,demo2,demo3,c="blue")

: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1a13fa4908>
```



IMPLEMENTATION OF DATA MINING SECURITY TECHNIQUES: One of the Key points that has be considered while working on data is the user privacy. Since the given data does not contain any private information about the user or any Social Security information, there is no major threat of data security of the user. Confidentiality and integrity of user data has been all through the project work. However, the login, printer, machine, file access details of the user are all required by the organization's management to keep track of the employees, and as this information is not as sensitive as credit card or other private information data mining security is not of much concern with respect to the given data in our personal opinion.

CONCLUSIONS:

After careful analysis of data using different data mining techniques we have come to the following conclusions about the user behavior based on the Login Access pattern, File Access, Printer Access, Program Access, email usage and machine usage.

Login Access Pattern:

1) From the type 1 data we found out that Users U01, U02, U03, U04, U05, U06, U07, U08, U09, U16, U18 and U19 are logging in to allocated Machines where as other users are logging into multiple machines.

2)all the Users have logged in to the machines on 22 days in the given month counting both weekdays and weekends, except for Users U18 and U19 who logged in for only 8 days in the given month.

3) User's U01, U03, U05, U07, U09, U11 logged in at a particular time interval (08:00 AM) and logged out (17:00 PM) at a particular time interval.

4)Considering 8-17:00 as working hours, User's U01, U03, U05,U07,U09, U11 has never logged in/out before or after working hours.

Program access Pattern:

1) UP350 program was executed maximum number of times.

2)From the Type 2 User data we observed that the UP type of programs are executed maximum number of times.

3) From the additional attributes we derived, it is apparent that most number of programs are executed during working hours and on weekdays.

4) Program LP075, LP085, UP095 are executed by U09 alone.

File access

1) From the type 2 user data we observed that F0200 is accessed by maximum number of users.

2) F0010, F0019, F0020, F0025, F0170, F0185, F0270, F0475 are having Read Write permissions

3) Files F0059, F0079, F0085, F0099, F0100, F0111, F0159, F0200, F0222, F0277, F0285, F0300, F0333, F0337, F0385, F0389, F0444, F0447, F0471, F0555 are having Read Status.

Printer access

1)From the type 2 user data we observed that Printer PR 2 is having high frequency and used by maximum number of Users (M05, M06, M07, M08, M09, M10, M21, M23, M26, M27, M28, M29, M30)

2) Printer PR5 is accessed only by U18.

3) PR5 printer printed less pages.

Email access

1) From the type 3 user data, we observed that Users only receive an email from bob@xyz.com and they don't send any emails to bob.

2) User's send email to jones and xyz but they don't receive email from jones and xyz.

3) Users can send and receive email from mom and smith.

4) There are no attachments while sending or receiving email from mom.

Machine Usage pattern

1) From the type 1 User data we observed that M18 and M19 were accessed even on weekends.

2)There are a total of 27 unique machines among which 15 are accessed by specific users and 12 are shared between users.

3) Machines M08 and M19 are used for maximum time and Machines M18 and M19 are the only two machines that are used during weekends in the given month.

4)Machine M08 is used by U08 and U10.